International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys.13, No.1 (Apr-23)

http://dx.doi.org/10.12785/ijcds/130163

Impact of Varying Strokes on Recognition Rate: A Case Study on Handwritten Mathematical Expressions

Sakshi[1], Vinay Kukreja[2] and Sachin Lodhi[3]

[1]*Chitkara University Institute of Engineering and Technology, Chitkara University, Punjab, India.*
[2]*Chitkara University Institute of Engineering and Technology, Chitkara University, Punjab, India.*
[3]*University Institute of Technology, Barkatullah University, Bhopal, Madhya Pradesh, India.*

Received 15 Mar. 2022, Revised 21 Jul. 2022, Accepted 19 Dec. 2022, Published 16 Apr. 2023

Abstract: Strokes constitute major units that form the contents of any handwritten text. When every user feeds or writes any handwriting sample, we archive strokes depending upon the context. In this study, the authors have experimented with recognizing handwritten mathematical expressions on datasets with varying strokes. The case study on mathematical expressions has been conducted where distinct datasets are analyzed, identified, and compared. A deep neural network-based recognizer has been deployed to test the formulated hypothesis around stroke characteristics. One experimented dataset had handwritten samples that had uniformly thin width of the stroke, whereas the other had strokes of varying width. After the experimentation, it has been observed that the dataset with thin stroke width (up to 1px) results in a significantly less effective recognition rate. The one achieved on experimenting with a dataset with varying strokes outcomes a jaw dropping recognition rate compared to the previous case. A comparative analysis has been performed to infer and affirm the hypothesis about stroke characteristics and their impact on the recognition rate.

Keywords: Strokes, Convolution Neural Network, Handwritten Mathematical Expression, Math Symbols, Math Expressions, classification

1. Introduction

Given the ubiquity of handwriting recognition, the idea of recognizing handwritten text with a more friendly human interface has been exclusively expounded [**?**]. The recent trends witness the thought of computing, where the computers could embed in the environment itself and make an easier way for human accessibility and transactions [**?**]. Specifically targeting mathematical expressions, the transaction idea becomes more exciting and challenging because of inherited complexities and ambiguities of symbols and signs [**?**]. Inputting mathematical text in systems through input devices like the keyboard calls for arduous efforts. The advent of advanced input mediums like touchpad and touch screenshots entirely transformed the trend of the existing input methodologies and made entering pretty smoother and more human-friendly [**?**]. The handwriting and its associated recognition tasks depend on various factors like internal handwriting characteristics (stroke width/size) or inherent external characteristics like dataset size. This study revolves around a case where the authors are endeavoring to measure the change or the difference around the elements of handwriting (strokes) and then infer the successful conclusions, targeting this internal characteristic of handwriting, precisely stroke width.

2. Background

A. *Defining Strokes*

A single stroke is a pen, stylus, or finger movement to generate a handwritten input or character. It actualizes the moment of the pen. The minimal unit forms any handwritten input strokes, sequences of points generated between pen-down and pen-up events at regular time intervals [**?**]. When collected in the form of a normal pen-ink, the strokes in their offline form are differently treated than the one which has been captured in the form of digital ink [**?**].

Figure 1a depicts a variable A, and Figure 1b highlights it is constituting strokes. 'A' variable here includes three strokes. Likewise, digit '2' is usually considered to be built in one stroke, and number '10' can be said to have been constructed from two strokes.

B. *Types of Strokes*

Depending on several parameters, we can classify the strokes. For instance, based on the mode of input or the ink with which it is penned, the type of script or writing is based on directions.

E-mail address: sakshi@chitkara.edu.in, vinay.kukreja@chitkara.edu.in, sachinlodhi8614@gmail.com

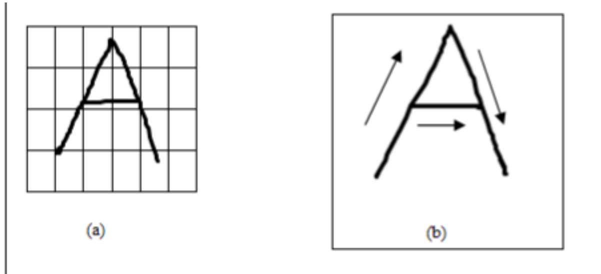

Figure 1. Stroke Illustration

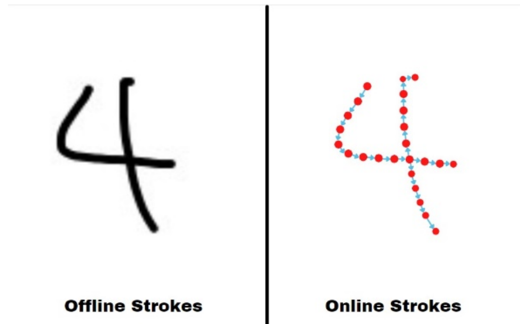

Figure 2. Offline versus Online Strokes

Based on mode of input mode, we have two types of strokes, one, i.e., penned with normal, natural pen ink, and the other includes strokes taken from digital ink. There is varied information in these inks, also called offline (pen-ink) and online strokes (digital ink), as shown in Figure 2. Based on the kind of scripts, we have two types of strokes; manuscript and cursive. Manuscript ones deal with digital inputs, whereas cursive ones are created around the hand drawings. Based on directions [?], we have eight directional strokes, namely Left to the right oblique line (direction: Upwards to the right), Horizontal lines (direction: Rightwards), Left to right oblique line (direction: Downwards to the right) Vertical line (direction: Downwards, Right to left oblique line (direction: Downwards to the left), Horizontal line (direction: Leftwards), Left to the right oblique line (direction: Upwards) to left (direction: Vertical line) as illustrated in Figure 3.

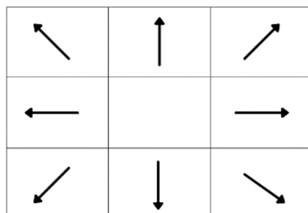

Figure 3. Types of Strokes (Based on directions)

3. Motivation and Objective OF THE STUDY

Strokes and their associated characteristics can sometimes prove to interestingly affect the recognition rate or accuracy of a recognizer that is built for handwriting recognition. Our primary objective and motivation for this work are to resolve and get more insights into how stroke sizes matter and influence the recognition model. The brief objectives of the study are listed as:

- To study and analyze the impact and difference the stroke characteristics accrue to the recognition process.
- To scrutinize and comprehend how the accuracy of the model varies with the variance of strokes that are used for training and testing.
- To test and get insights about the stated hypothesis about strokes and add a new dimension to ongoing research in the community of pattern and handwriting recognition.

4. Preview of Literature: the state of art

Mathematical symbol and expression recognition has persisted in being a topic of research which has embarked to be researched in 1967 [?], but the current decade witnesses the active attention and exploration by the research community [?], [?]. In the works for offline mathematical expressions, Chan [?] developed a stroke extraction algorithm. The authors made use of binarization, skeletonization, breakdown into segments, stroke reconstruction, order normalization, and other procedures are included to construct this method. As the authors points out, offline to online transition methods necessitate retraining the online recognition model with extracted strokes. Another study targeting online handwritten mathematical text is experimented by Zhang [?] where the researchers addressed the problem and challenges related to handwritten mathematical expression recognition by implementing tree-based BLSTM architectures. The authors lead the inputs strokes to direct labeling of nodes that held the symbols and edges, including relationships from the developed graph modeling. There were no instances of explicit segmentation or recognition, or grammar structures involved in this architecture. There are recorded two important phases in mathematical expression recognition; symbol recognition and structural analysis [?], [?], [?].

The symbol recognition process has always been considered to be crucial. The study by Feng [?] proposes a squeeze-extracted multi-feature convolution neural network (SE-MCNN) to enhance the recognition rate of handwritten math symbols. The system proposed in this paper integrates the eight-directional feature of the original sequence in the convolutional layer, which significantly compensates for the lost dynamic trajectory information. Another study presents a system for recognizing online handwritten mathematical expressions (MEs) by applying improved structural analysis. With the proposed system, MEs are represented in the form of stochastic context-free grammar (SCFG), and the Cocke–Younger–Kasami (CYK) algorithm is used to parse two-dimensional (2D)

structures of online handwritten mathematical expressions. To the best of our knowledge, no study has focused entirely on stroke-based characteristics and interesting limelight conclusions from it. Our article is completely focused on this aspect, and the investigation has introduced new and novel conclusions in this context. Handwritten mathematical expression recognition has recently seen significant progress in encoder-decoder models. Study experiments show that our model improves the ExpRate of current state-of-the-art methods on CROHME 2014 by 2.23 percent when compared to methods that do not use data augmentation. On CROHME 2016 and CROHME 2019, we increase the ExpRate by 1.92 percent and 2.28 percent, respectively [?]. Impressive results have been achieved thanks to HMER's encoder-decoder architecture. For HMER, the author came up with a simple and effective encoder-decoder network that includes syntax information. The author's grammar rules are used to convert each expression's LaTeX markup sequence into a parsing tree, and a deep neural network is used to model the markup sequence prediction as a tree traverse process. Our method outperforms previous efforts in recognition accuracy when tested on three benchmark datasets. In order to prove the efficacy of our method, we gathered 100k handwritten mathematical expression images from ten thousand writers [?].

5. Implementation

A. Dataset

Our case study has targeted two types of datasets for analyzing the impact of stroke on the recognition rate. One dataset includes mathematical expressions with strokes of thin width, i.e., 1px, whereas the second includes mathematical expressions of varying stroke width. The descriptive details of dataset-1 and dataset-2 are as follows:

1) Dataset-1

The dataset has been downloaded from kaggle [?]. The dataset comprised of 375974 jpg files (45×45px). Basic Greek letter symbols such as alpha, beta, gamma, mu, sigma, phi, and theta are included. There are also occurences of alphanumeric symbols in English. All symbols are contained in 82 symbol classes. The total Set operators, all math operators, basic math functions like log, lim, cos, sin, and tan. Other math symbols include int, sum, sqrt, delta, and others. List of symbols included are ['forall', 'sin', 'prime', '!', '2', ',', 'lim', 'H', 'theta', 'N', 'gt', 'in', 'gamma', '0', 'ldots', 'Delta', 'z', 'y', 'cos', 'mu', 'j', 'p', 'k', 'C', 'M', 'beta', 'sum', 'G', '{', '8', 'exists', 'forward_slash', '6', '3', 'e', 'log', 'i', 'pm', 'div', '[', '-', 'rightarrow', 'sigma', 'leq', 'w', 'geq', '9', 'sqrt', 'f', 'times', 'u', '5', 'X', 'A', '4', 'pi', 'd', '1', 'infty', 'q', ']', 'ascii_124', 'R', 'o', 'phi', ')', 'T', 'alpha', '7', 'lt', '}', '(', 'int', 'v', 'b', 'S', '+', 'lambda', 'l', 'neq', '=', 'tan']. The glimpse of sample images from the dataset are as shown in Figure 4.

2) Dataset-2

Over 9000 images of handwritten digits and arithmetic operators are included in this dataset downloaded from Kaggle [?], and the total number of symbol classes is 19. All the digits from 0-9 are part of this dataset, and the basic maths operators like plus, minus, multiplication, decimal, division, and equals are also included. The majority of the images have a resolution of 400×400 pixels. Some could be 155×155. Each class has approximately 500 examples. The dataset incorporated has symbols like ['2', 'add', 'eq', '0', 'dec', 'x', 'sub', 'z', 'y', '8', '6', '3', 'div', '9', '5', '4', '1', '7', 'mul']. A glimpse of the sample images from the dataset is shown in Figure 5.

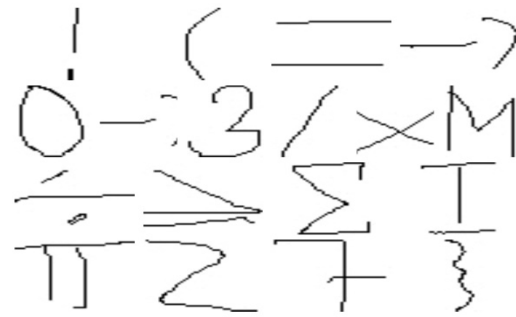

Figure 4. Glimpse of samples

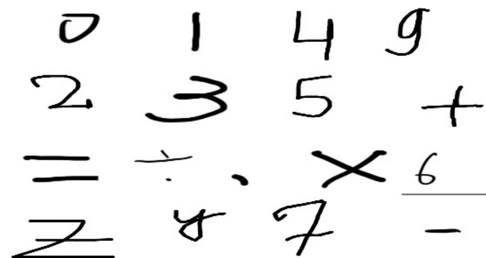

Figure 5. A glimpse of samples from dataset-2

B. Experimental Setup

The experimentation has been carried out on the system with disk space specifications of 108 GB and internal memory of 12.0 GB. The NVIDIA GeForce MX 130 based on two Kepler GPU has been used in the process. The Ubuntu 20.04.2 LTS is the OS installed on the implemented system, 64 bits. The Intel ® core ™ i5-8250U processor with eight logical processors and four cores is the built-in processor of the method used for experimentation. The implemented code construct for our acquired module has been executed on Google Collab, its in-built GPU. The google collab configuration is based on Intel® Xeon CPU having a clock rate of 2.20 GHz, a disk space of 108 GB, and a memory limit of 12.69 GB. The NVIDIA Tesla K80 GPU, which is 12 GB, has been deployed in the process.

C. Data Preprocessing

1) Grayscale

The considered samples from the dataset need to be grayscaled where the set of images that are in different colors like RGB, HSV, etc. can be transformed in the

range of pixel values (0-255). This preprocessing of the samples from the dataset is essential to reduce the dimension and accomplish the channel reduction process where colors channels adding complexity to the model could be transformed in varying shades between black and white, as shown in Figure 6.

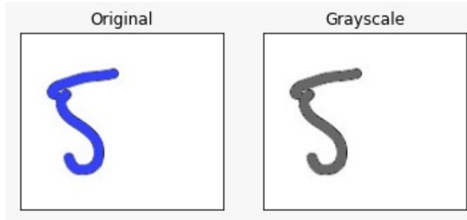

Figure 6. Original image transformed to grayscale (Preprocessing layer)

2) Binarization

The transformed grayscale image needs to be converted into a bi-level document image. Here, the pixels of varying intensities are binarized into vector entities. To remove the noise from the image, this step of binarization becomes crucial to be performed. At times binarization is performed against some threshold value, where the pixel intensities above the specified threshold values are binarized into 1. The authors used the threshold() function to accomplish the process of binarization. The output is shown in Figure 7.

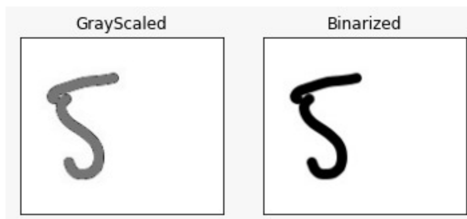

Figure 7. Grayscaled into Binarized Image (Preprocessing layer 2)

D. Data Augmentation

To avoid the chance of overfitting, the authors performed data augmentation by deploying methods like rescaling, shearing, and zoom. This process is mostly about changing and bringing such modifications in the images, which can benefit this machine learning experiment in two ways. One by diminishing the chances of overfitting (an issue in machine learning models), and another by enlarging the dataset size. Rescaling is one of the layers of preprocessing that rescales input values to a new range. This layer rescales each value of the input (typically an image) by multiplying by scale and summing offset. The sample code construct is presented below:

$$ImageDataGenerator(rescale = 1./255, shear_range = 0.2, zoom_range = 0.2)$$

Where the value 1/255 is the scaling factor, and the specified shear value is 0.2 which denotes the factor of shearing or angle of shearing. Shearing allows the image distortion along both the x and y-axis. It directs the model of how humans view and visualize from different angles and perspectives. The zoom range for augmentation purpose have been specified as 0.2 which would eventually give us zoom factor in range of [0.8,1.2]. This would generate more images within the zoom range of [0.8, 1.2]. This would help the model to understand the image from a different distance.

E. Recognition Model

1) Splitting

Both the experimented datasets are split into training and testing sets. In the case of dataset-1, 75%, i.e., 282007 images belonging to 82 symbol classes, are assigned to training split, and 25%, i.e., 93967, are allocated to test split. The target size of the image is 45×45, i.e., each image in the dataset is 45 px × 45 px. The color mode has already been initialized to grayscale. For dataset-2, as per the same ratio of training and testing, 7560 images belonging to 19 symbol classes of dataset-2 has been used for training, and 2511 images of the dataset have been used for testing purpose.

2) Preparing output classes

To test on real-time data, the classes of the symbols are prepared from the directory(s) name given in the dataset. The list prepared for each dataset would be iterated over, and the symbol(s) in image form would be read and sent to the recognizer. The list of classes from the first dataset would be of length 82, i.e., the number of the symbols in the first dataset would be 82. In the same manner, the prepared list from the second dataset would contain 19 classes. Both the list would be used to extract images from the real-time testing data by performing iteration over these prepared lists of symbols.

3) Model Initialization

The model has been initialized using a sequential function of the Keras library. This Sequential model API is a method of creating deep learning models that involve building an instance of the Sequential class and adding model layers to it. Machine learning models that input or output data sequences are called sequence models. Text streams, audio clips, video clips, time-series data, and other types of sequential data are examples of sequential data. Images are the data stream.

$$model = tf.keras.models.Sequential()$$

4) Construction of Recognizer (Adding layers)

The constructed neural network-based model (Figure 8) constitutes several layers in which the first one is the first convolution block. Thus, the authors added the Convolutional2D layer, and above it, the authors embedded the added the MaxPool2d layer for downsampling. In the first or first Conv2D layer, there are 32 filters, and the kernel size is 6x6. Input shape is initialized by 45 × 45 with a single channel. An activation function here deployed is relu in the first block. In the first MaxPool2D layer, the

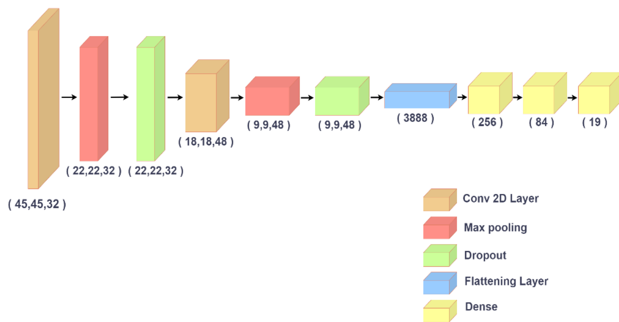

Figure 8. Core Architecture of Recognizer (Part of Model)

stride value is 4. In the second block, the authors added another Conv2D layer with 48 filters, kernel size of 5 × 5, and relu activation. Thereafter, the authors stacked another MaxPool2D layer with a stride value of 3. In the last block or classifier head, the authors first reduced the dimensions to 1D (one-dimensional array) by adding a flattening layer. After that, the authors performed the stacking of 2 Dense layers with 256 units and 84 units. The last block consists of fully connected layers. The dimensionality is reduced to the number of classes in the output layer, i.e., 25. The output shape and number of parameters are shown in Table I.

5) Integration with tensorboard (visualization)

The authors initialize the tensorboard module to integrate it with the model training phase. This would be used for post-visualization of the training on both models. Tensorboard gives various graphs like accuracy vs. epoch, epoch vs. loss, iteration vs. accuracy, and many more. All this result-based analysis can be visualized by specifying and integrating well with the tensorboard module of Python.

6) Optimizing Decision Parameters

Adam optimizer has been deployed with learning rate as specified through the code segment as mentioned below:

$$adam = tf.keras.optimizers.Adam(learning_rate = 5e - 4).$$

The difference between the real value and predicted value constitutes the loss. Here, the loss is specified as loss = 'categorical_crossentropy'. Here Categorical cross-entropy is a loss function used in multi-class classification problems. These are tasks in which an example can only belong to one of many possible categories, and the model must discern which one it is. Its formal purpose is to quantify the difference between two probability distributions. To save the log of training, the authors also initialized the log directory. In addition to that, a tensorboard object is created to read the log data from the directory for visualization. Further, the next step is to compile the model.

7) Compilation

After initializing all the model parameters, the authors execute the compile() and generate the compiled version of the model, ready to be trained. It also checks and validates for format errors and saves the debugging time and launch of the model.

8) Training

Using model.fit(), the authors start training the model for 20 epochs, and the logs of the entire training would be saved in the callback, i.e., initialized tensorboard object.

a) Model-1: Training with dataset-1

Before finalizing 20 epochs, the model has been trained and scrutinized for 5, 10, 15, 25 epochs earlier. It has been observed that achievable accuracy with five epochs is found to be 98.26%, and with 10 epochs, it was estimated as 96.82%. On running with 15 epochs, the accuracy achieved is 99.17%. While executing with 20 epochs, the accuracy is measured as 99.25%, and with 25 epochs, 97.27%. On experimenting with 25 epochs, the model training time got extended. The authors also realized that there could be chances of overfitting at 25, as the accuracy also dropped out at this instance.

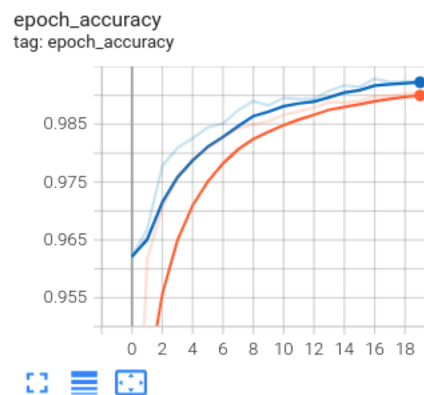

Figure 9. Epoch versus Accuracyfor Model 1

With every epoch, there have been observed varying accuracies. The resulting accuracy up to the considered epoch range has been mapped in Figure 9, which vividly depicts the exponential rise of accuracy values up to epoch 20. The blue line in the graph depicts the training trend, whereas the orange/red line in the graph is for the validation trend. The metrics such as accuracy are extracted parallel for training and validation phases. Though the model further may be tested by the developers for varied datasets, the model itself extracts the accuracy for validation known as evaluation accuracy, which can be further compared with accuracy achieved on testing. Hence, this can be used for cross-validation.

Loss is the difference between the ground truth and the prediction of the network. It is desirable to have fewer values for loss. Therefore, with every epoch, the extracted loss values are mapped in Figure 10, which lucidly indicates the loss function to be considerably diminishing.

The evaluation accuracy extracted as the result of the self-validation characteristic of the developed model is here mapped with iterations. Iteration is the processing of every batch of the dataset. Evaluating accuracy is present on the

TABLE I. Basic Layer Details of the model

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 45, 45, 32)	832
max_pooling2d_1 (MaxPooling2D)	(None, 22, 22, 32)	0
dropout_1 (Dropout)	(None, 22, 22, 32)	0
conv2d_2(Conv2D	(None, 18, 18, 48)	38448
max_pooling2d_2 (MaxPooling2D)	(None, 9, 9, 48)	0
dropout_2 (Dropout)	(None, 9, 9, 48)	0
flatten_1 (Flatten)	(None, 3888)	0
dense_1 (Dense)	(None, 256)	995584
dense_2 (Dense)	(None, 84)	21588
dense_3 (Dense)	(None, 19)	1615

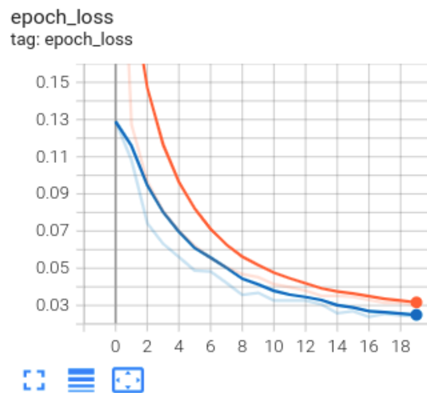

Figure 10. Epoch versus Loss for Model 1

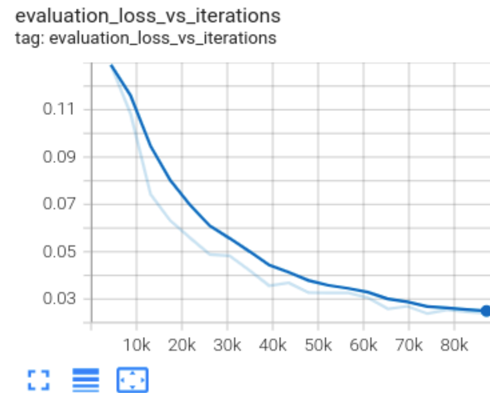

Figure 12. Evaluation loss versus iterations for Model 1

y axis of the graph presented in Figure 11, and the x-axis denotes the number of iterations. As the iterations increase, the model keeps on learning more features from the inputted dataset; thus, the evaluation accuracy keeps on escalating, as depicted in Figure 11.

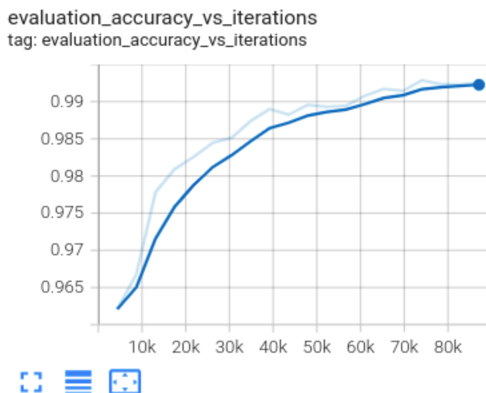

Figure 11. Evaluation accuracy versus iterations for Model 1

Similarly, evaluation loss has been mapped for every iteration where the loss seems to be highly dwindling, as shown in Figure 12.

b) Model 2: Training with dataset-2

Before finalizing 20 epochs, the model has been trained and scrutinized for 5, 10, 15, 25 epochs earlier. It has been observed that achievable accuracy with 5 epochs is found to be 87.81%, and with 10 epochs, it was estimated as 92.39%. On running with 15 epochs, the accuracy achieved is 92.87%. While executing with 20 epochs, the accuracy is measured as 94.34%, and with 25 epochs, 93.16%. The authors chose 20 epochs for training as the instance 25 epochs witnessed the probability of overfitting in this case. Also, the training time accuracy of the model is found to be attenuated.

Similarly, the same training analysis graphs are plotted for model-2, which has been trained on the dataset with varying strokes. It has been observed that the model witnesses changing accuracies with every epoch. The resulting accuracy up to the considered epoch range has been mapped in Figure 13, which vividly depicts the exponential rise of accuracy values up to epoch 20. The blue line in the graph depicts the training trend, whereas the orange/red line in the graph is for the validation trend. The shadows that can be noticed in the background (Figure 13) are the actual mapped values that are further smoothened to bring up a consolidated value range in the form of blue and red lines that apparently depict the accuracy trends for training and

validation, respectively.

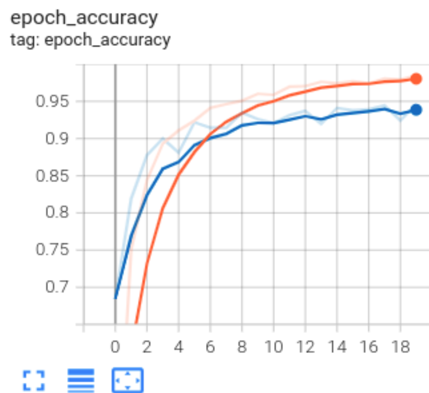

Figure 13. Epoch versus Accuracyfor Model 2

Similarly, the loss noticed during training is mapped with the epoch in Figure 14. Also, the evaluation accuracy (accuracy noticed as part of model's validation) and evaluation loss has been mapped with the iterations in Figures 15 and 16.

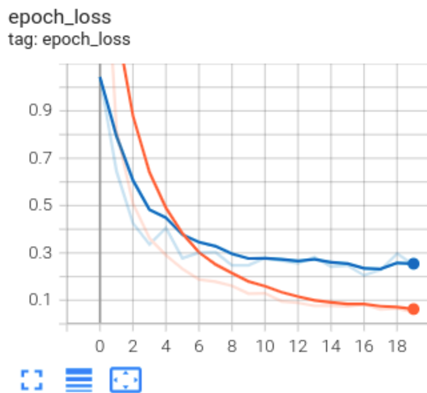

Figure 14. Epoch versus Loss for Model 2

9) *Testing on unseen real-time data*

The models trained on different datasets are again tested for the unseen real-time dataset to observe and notice comparative differences between them (if any) existed.

a) *Testing Analysis on Model-1 (Trained on thin stroke dataset-1)*

The recognition model-1 trained on dataset-1 (with thin stroke width) is tested on real-time data. In the case of testing, a few symbols fed to the model are part of the expression, and the accuracy achieved for individual recognition of inputted symbols is listed in Table II.

Table III illustrates the recognition rate per each symbol that has been correctly identified. The first row holds equals symbols whose 2859 images have been corrected identified

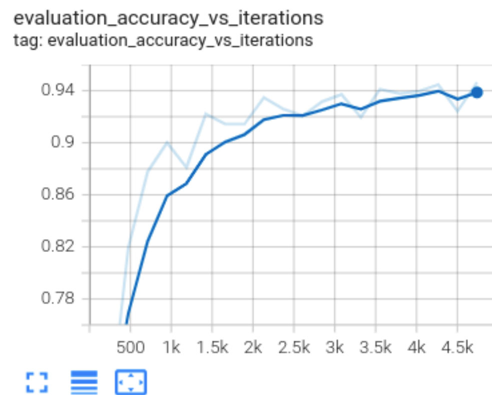

Figure 15. Evaluation accuracy versus iterations for Model 2

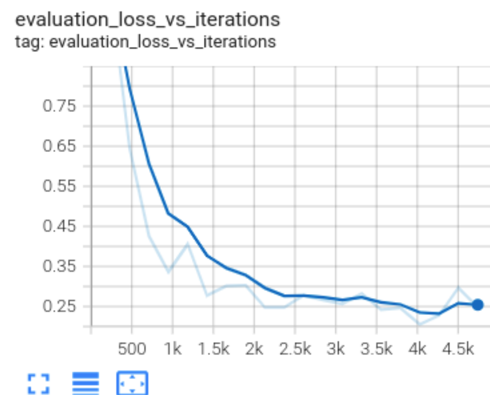

Figure 16. Evaluation loss versus iterations for Model 2

from the set of 8368. The following formula then evaluates the absolute accuracy.

$$Average\ Accuracy\ of\ Model = (total\ correctly\ recognized\ images) / (total\ inputted\ images)$$

Thus, as the total correctly recognized symbol images are 2859, and the total symbol images fed to the model are 8368, an accuracy of 34.01% has been achieved.

b) *Testing Analysis on Model-2 (Trained on varying stroke dataset-1)*

The recognition model-2 trained on dataset-2 (with varying stroke width) is tested on real-time data. In the case of testing, a few symbols fed to the model are part of the expression. The accuracy achieved for individual recognition of inputted symbols is listed in Table III.

Table II illustrates the recognition rate per each symbol that has been correctly identified. The first row holds equals symbols whose 1775 images have been corrected identified from the set of 2002. The absolute accuracy is then evaluated by the formula for accuracy of the model

TABLE II. Accuracy of Model-1 on symbols from the unseen dataset

Symbols	Correct/Total	Accuracy
z	145/668	21.7
eq(=)	334/631	52.9
5	89/270	32.9
4	354/467	75.8
9	88/375	23.46
add(+)	408/568	71.83
div	116/558	20.78
y	54/638	8.46
0	46/470	9.78
3	166/415	40
2	56/345	16.23
mul	176/538	32.71
sub	439/626	70.12
6	103/424	24.29
8	58/421	13.77
7	134/466	28.75
1	93/488	19.05
Accuracy	2859/8368	34.01

TABLE III. Accuracy of Model-2 on symbols from the unseen dataset

Symbols	Correct/Total	Accuracy
eq(=)	1775/2002	88.66
5	1605/2002	80.17
4	1632/2002	81.52
9	1242/2002	62.04
add(+)	1894/2002	94.61
div	380/869	43.73
y	1275/2002	63.69
0	1865/2002	93.16
3	1475/2002	73.68
2	800/2002	39.96
mul	1921/2002	95.95
sub	1981/2002	98.95
6	1919/2002	95.85
8	1829/2002	91.36
7	982/2002	49.05
1	939/2002	46.09
Accuracy	23514/30899	76.09%

mentioned in 4.4.9.1.

Thus, as the total correctly recognized symbol images are 23514, and the total symbol images fed to the model are 30899, the authors achieved an accuracy of 76.09%.

F. Results and Discussions (Comparative analysis)

The accuracies achieved for individual symbols by Model-1 and Model-2 have been compared and plotted in Figure 17. The figure depicts that Model-2 (trained on strokes of varying width) exhibited significantly effective accuracies than the model-1 that is trained on strokes of thin width. It is also an observation that Model-2 (trained on strokes of varying widths) have extracted better features than the model trained on thin stroke width. The blue bars denote Model-1, and oranges ones depict the accuracies exhibited by Model-2.

There have also been observations around the symbols that gave considerably fewer accuracies like '÷' was confused with '+', 'z' was misidentified with '2'. '1' was misinterpreted with '!', '0' was misperceived with 'o', '=' when not in proper alignment was miscategorized into some other symbol classes.

G. Conclusion and Future Goals

This paper revolves around noticing the impact and influence of stroke characteristics on the accuracy of the recognizer. The authors have considered a deep neural network-based recognizer and tested it upon two types of

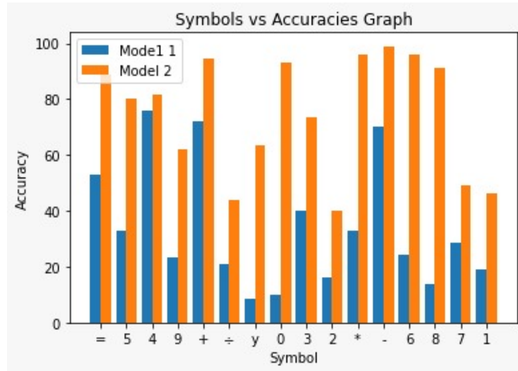

Figure 17. Comparative Analysis of Accuracies of Model-1& Model-2

datasets. The one with a thin stroke width of as much as 1px, whereas the other dataset included comparative more comprehensive stroke and some varying width strokes. It has been noticed that recognition accuracy has been tremendously improved while experimenting with strokes of wider width. When tested on unseen data, the model that was trained on Dataset-1 (thin strokes) achieved an accuracy of 34.01%. The contrary model trained on dataset-2 (varying strokes) resulted in an accuracy of 76.09% when tested on real-time unseen data. Based on training logs, it can be presumed that the first model trained on thin strokes extracts most of the features from the dataset during the very first few epochs and the problem of overfitting has a decent possibility. Our future goal is to extend this case study on strokes of distinct pixels texture and include several other complex datasets. Experimentation has the scope to alter and experiment with layers of the deployed neural network-based recognizer. Also, the symbols causing ambiguity could be handled with special attention.

Sakshi is currently a full-time Ph.D. student at Chitkara University, Punjab. She has completed her Msc. in Computer Science at Guru Nanak Dev University, Punjab. Her area of interest is pattern recognition, machine learning, and deep learning. She has published more than 10 papers in reputed journals and conferences. She is deliberately attending and presenting papers at several national and international conferences.

Vinay Kukreja earned his Ph.D. Degree in computer science from Chitkara University. He is presently working as a Professor at Chitkara University, Punjab, India. He has more than 16 years of teaching experience; has published more than 55 national/international papers in reputed journals & conferences, 3 books, granted 3 patents and filed more than 20 patents.

Sachin Lodhi has completed his BTech in 2021 from UIT, Barkatullah University, Bhopal. He has been working on browser automation for two years. His core research areas include image processing, image analysis, automation, computer vision, machine learning and IoT. He has in-depth and hands-on knowledge in the field of Artificial intelligence and Image Processing. He has done many projects.