Article

Handwritten Character Recognition on Android for Basic Education Using Convolutional Neural Network

Thi Thi Zin [1,*], Shin Thant [1], Moe Zet Pwint [1] and Tsugunobu Ogino [2]

[1] Graduate School of Engineering, University of Miyazaki, Miyazaki 889-2192, Japan; shinthant3010@gmail.com (S.T.); smithmoezet01@gmail.com (M.Z.P.)
[2] KJS Company LTD., Miyazaki 880-0001, Japan; t-ogino@e-kjs.jp
[*] Correspondence: thithi@cc.miyazaki-u.ac.jp

Abstract: An international initiative called Education for All (EFA) aims to create an environment in which everyone in the world can get an education. Especially in developing countries, many children lack access to a quality education. Therefore, we propose an offline self-learning application to learn written English and basic calculation for primary level students. It can also be used as a supplement for teachers to make the learning environment more interactive and interesting. In our proposed system, handwritten characters or words written on tablets were saved as input images. Then, we performed character segmentation by using our proposed character segmentation methods. For the character recognition, the Convolutional Neural Network (CNN) was used for recognizing segmented characters. For building our own dataset, handwritten data were collected from primary level students in developing countries. The network model was trained on a high-end machine to reduce the workload on the Android tablet. Various types of classifiers (digit and special characters, uppercase letters, lowercase letters, etc.) were created in order to reduce the incorrect classification. According to our experimental results, the proposed system achieved 95.6% on the 1000 randomly selected words and 98.7% for each character.

Keywords: handwritten character recognition; basic educational application; offline self-learning application; projection; closed character detection; cursive character recognition; Convolutional Neural Network

check for updates

Citation: Zin, T.T.; Thant, S.; Pwint, M.Z.; Ogino, T. Handwritten Character Recognition on Android for Basic Education Using Convolutional Neural Network. *Electronics* **2021**, *10*, 904. https://doi.org/10.3390/electronics10080904

Academic Editors: Andrzej Czyżewski, Piotr Szczuko and Giovanni Dimauro

Received: 1 March 2021
Accepted: 7 April 2021
Published: 10 April 2021

1. Introduction

Education is very important to human life. Education enables people to have a quality life by gaining knowledge and building character. Education provides a broad-based view of life, instilling ideals and teaching adaptability to a changing environment. Therefore, all citizens have the right to a quality education [1]. However, according to the UIS global data for the school year ending in 2018 [2], over 59 million children of primary school age were not in school. Most of the children in rural areas often sell goods on the street or work in the fields to supplement family income, as seen in [3]. The author also claims that students often walk more than 3 km to and from their schools every day since few schools are in rural areas. Moreover, the lack of resources and an insufficient number of teachers present major barriers to a quality education. How can we provide high quality education for all children equally? For these reasons, we proposed an Android application for primary education. Using this application, children can study anytime, anywhere. Transportation also will not be a problem anymore. The courses are developed according to quality standards. Since the application includes an automatic checking system, students can practice alone. The application is not only useful in rural areas, as children confined to their homes for any reason can get an education. Parents can arrange for their children to study at home without anxiety.

Research findings in [4] indicated that the advantages of using handwriting applications along with traditional teaching for children can improve handwriting skills. The benefits and drawbacks of mobile learning are discussed by the authors in [5]. Many applications have been developed to improve basic education [6]. Most of these are designed to teach vocabulary by matching words with images. However, they do not help children memorize spelling. Some provide a virtual keyboard for memorizing spelling, but they do not provide enough tips for memorizing writing. Some provide traces for writing characters but cannot effectively teach spelling. We developed an application using image processing concepts to overcome those problems. Our application can support children in practicing writing and efficiently learning the spelling of English vocabulary. Since the application also provides an immediate check of written words, children can study without an experienced teacher. Children can practice English vocabulary, basic mathematical operations, and special characters through our system. The system is designed to recognize 72 characters, including the 26 uppercase English letters (A–Z), the 26 lowercase letters (a–z), 10 digits (0–9), 8 basic mathematical symbols ($+$, $-$, $\times$, $\div$, $=$, $>$, $<$ and . (the decimal point)), and two more special characters (? and !). Since primary-level students often write uneven, skewed or slanted characters, schools use ruled books with four lines to control their writing. Moreover, the uneven characters can cause some unexpected data errors. Our application provides four lines on the drawing area not only to make the application child friendly, but also to help them practice their writing with standard sized letters.

When they write down and submit a word, the system saves the written answer as an image, performs a step-by-step process on the saved image, and finally, provides the recognition result for the written characters. The application is developed in the simplest and most efficient way. Character segmentation is performed using basic image processing concepts. We label connected objects, combine labels, and then separate labels before segmentation with the projection profile method. Since even the same characters vary in shapes, we cannot classify characters with simple image processing techniques. So, we need to use effective machine learning algorithms.

The authors described an exhaustive review and updated survey with state-of-the-art methods on two well-known datasets, MNIST and EMNIST in [7]. These two datasets are broadly used in the area of handwritten character classification and recognition. To propose an offline handwritten Javanese character recognition, the authors in [8] used image processing methods for character segmentation and the Convolutional Neural Network (CNN) model to build recognition software. For the performance evaluation, the authors compared their proposed model with multilayer perceptron (MLP) models, including classification accuracy and training time. The Tamil handwritten character recognition is proposed in [9] by using image pre-processing steps, and then followed by CNN for recognition. The authors also introduced a new Self Adaptive Lion Algorithm (SALA) to fine-tune the fully connected layer and weights in CNN. For the optical character recognition task, the authors in [10] proposed Dense Residual Network (DRN) which is a combined structure of residual dense block (r-RDB) and global dense block (GDB) to capture both local and global features.

In our proposed system, we used a Convolutional Neural Network (CNN) for character recognition. In the traditional machine learning model, feature vectors are input into the machine learning algorithm. The feature extraction and feature reduction processes are needed to perform as the primary step. On the other hand, deep learning models, such as CNN, only are needed to provide the input image. The feature extraction and classification processes are automatically done by the CNN framework. Our proposed CNN model is simply composed with three convolution layers and two fully connected layers. In order to reduce the workload on the tablet operating system, the network model is trained on a high-end machine. Training and testing data are collected as traditional handwriting from primary level students from four different countries: Bangladesh, Myanmar, Nepal and the Philippines. We also collect some handwritten data from Myanmar students through a simple tablet application. Since the system is developed to recognize both alphabetical letters and digits, different types of classifiers are created to reduce some conflicting results.

Our application focuses on accurate writing styles because the objective is to practice writing English. Therefore, our approach is to use simple character segmentation and recognition, so our proposed system might be less applicable for complex writing styles. However, we can provide an effective environment for students to practice written English compared with other educational applications. The overall system of our proposed work is summarized as follows:

- In the character segmentation part, we proposed the simple concepts of labeling and projection to overcome the segmentation problems that are found after initial seg-mentation.
- Since the network model is trained on a high-end machine and converted into a very light classifier which will be deployed on mobile devices, therefore, the model can be loaded quickly for each character classification process.
- The application provides an effective learning environment without requirements for experienced teachers or internet service. It can be accessed anytime, anywhere. The standard courses provide equally high quality for all users.

The rest of this paper is organized as follows. In Section 2, we review some related research on handwritten character segmentation and recognition on an Android system. Section 3 covers the steps of our proposed methods. The experimental results and discussion of the results are described in Section 4. The paper is concluded, and future work is discussed in Section 5.

2. Related Work

As a subset of optical character recognition (OCR), the technology for handwritten character recognition needs refinement, and is the subject of research worldwide. In recent years, OCR concepts have been used in applications for banking, healthcare, and the legal industry, e.g., for invoice imaging [11]. The authors in [12] and [13] apply various projection methods for the segmentation of digits and CNN for recognition. In another work, Kannada optical character recognition was implemented on the Android operating system for Kannada sign boards [14]. In the described research, a Kohonen network was used to recognize segmented characters. Other related research on recognizing English letters has also been carried out using a simple feature extraction and neural network [15]. A neural network was also applied to research on the conversion of English text into Marathi text. This research was also carried out on the Android platform. In this application, an image with English text is scanned with an Android camera, subjected to several image processing steps, recognized through a neural network, and then converted into Marathi text [16]. Not only for English text, but also for other languages such as Malayalam, a Convolutional Neural Network (CNN) is applied for character recognition in [17].

In Reference [18], the authors proposed a model by presenting Deep Convolutional Recurrent Network (DCRN) to recognize handwritten Japanese text lines. Their model is composed of three parts: the Convolutional Neural Network (CNN) and sliding window method, used for feature extraction from a handwritten image; Bidirectional Long short-term memory (BLSTM), applied for recurrent layers to make predictions; and finally, the Connectionist Temporal Classification (CTC) decoder, used for the transcription layer to make the recognition process. To extract textual information from medical laboratory reports, the authors in [19] proposed a system by using a deep learning approach. For text detection, Faster RCNN [20] combined with a patch-based training strategy is applied and Convolutional Neural Network with a concatenation structure is proposed for the text recognition part. For Arabic handwritten character recognition, the authors in [21] proposed a mobile application by using cloud computing and Google APIs. The handwritten recognition part is developed on a cloud computing platform with the approach proposed in [22]. Then, recognized text is returned to the mobile phone to apply in the Google map API for locating the Arabic address on the map, and the Google translate API for translating Arabic text to English text.

In our prior work, we mainly focused on the character segmentation part that can handle segmentation errors [23]. In this paper, we modified our prior segmentation method to solve the over segmentation errors. We also combined it with the character recognition part to develop the complete application. In the handwritten character recognition field, machine learning and deep learning algorithms play an important role. In Reference [24], the authors proposed a handwritten character recognition model by using deep neural networks and TensorFlow libraries. To perform specific handwritten character recognition on mobile devices, the lightweight network model is designed to perform a character classification process [25]. The authors also described the steps that are applied in the data collection process by using mobile devices.

Many useful applications can be developed by using OCR on the Android platform, including our proposed system. Our system is developed with our own dataset for an effective application area which will be applied as a teaching aid for children's education. Since OCR is more reliable and provides more accurate results with machine learning algorithms, we need massive amounts of training and testing data. Our proposed system performs a network training process on a high-end machine instead of loading massive amounts of training data on the Android platform. Using simple image processing methods without a deep learning approach provides efficient and reliable segmentation results in our system. Our system can also give good recognition accuracy by using a simple network with lightweight, learnable parameters, which is more suitable for mobile devices.

3. Proposed System

The handwritten image is segmented and recognized. In the segmentation phase, the input image is firstly pre-processed. Secondly, the connected components are labeled. Thirdly, the individually labeled objects are segmented using the vertical projection method because some words might be written cursively. Finally, some over-segmented parts are removed, and the image is segmented with the remaining segmentation points. Then, the resulting segmented characters are recognized with a Convolutional Neural Network (CNN) classifier. The classifier is trained on a high-end machine and changed into an Android compatible (.tflite) file using the TensorFlow model. Finally, the classification result for each character is shown to the user. An overview of our target application is shown in Figure 1 and the flowchart is shown in Figure 2. This includes preprocessing for input characters or words, segmentation, character classification and the checking process.

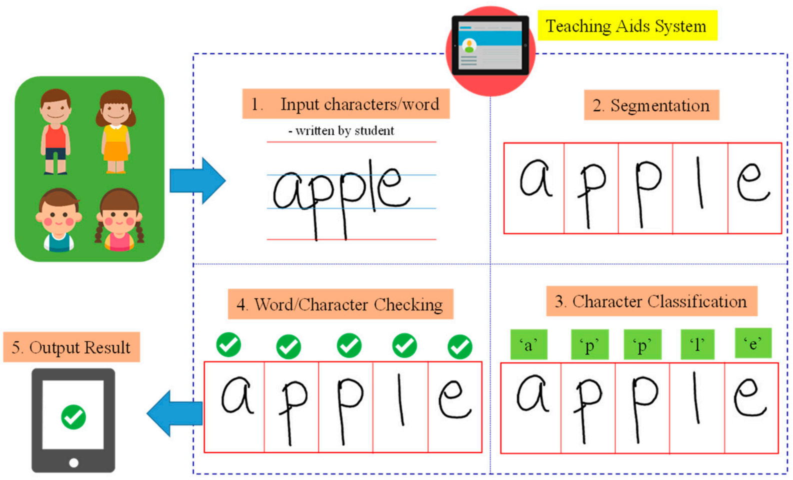


Figure 1. Overview of the target application.

In our current system, we do not work on the character- or word-checking process. In the word-checking section, the classified result is compared with the correct ground truth answer. When incorrect characters are found, these characters are highlighted. It will be combined in our future application.

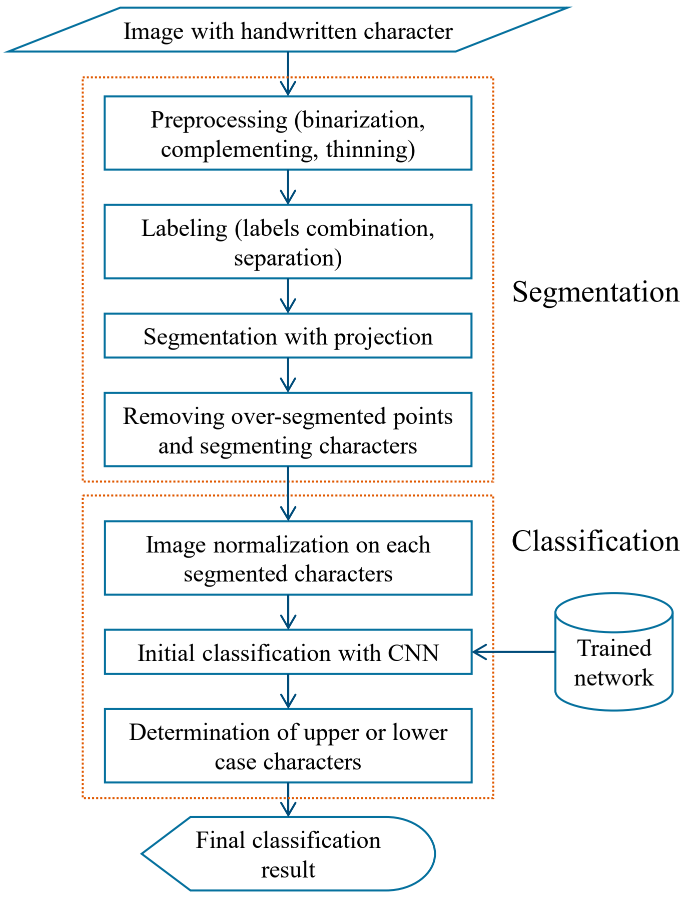


Image with handwritten character
Preprocessing (binarization, complementing, thinning)
Labeling (labels combination, separation)
Segmentation with projection
Removing over-segmented points and segmenting characters
Segmentation
Image normalization on each segmented characters
Initial classification with CNN
Trained network
Determination of upper or lower case characters
Classification
Final classification result

Figure 2. Flowchart of the proposed system.

3.1. Segmentation

Segmentation is the process of separating individual characters from the handwritten word. We perform pre-processing on the input image before extracting characters. We use a labeling process to get each connected object. Then, we combine some characters such as i, j and ?. Since some characters might be connected to each other, they can be assumed to be one character if we only use the labeling approach. Therefore, we apply the projection profile method to segment cursive characters.

3.1.1. Pre-Processing

In the pre-processing step, the four lines from the ruled screen in the saved image must be removed without affecting the handwritten word. Moreover, the saved image must also be binarized and inverted to facilitate subsequent steps of image processing. Therefore, we perform pre-processing by inverting black and white pixels. We invert all black pixels into 1 and the other pixels into 0. Although the background in processed images is black in our system, a white background is used in this paper for better visualization. Since vertical projection is used for segmentation, a morphological thinning process is performed on the image. Step-by-step pre-processing for the word “birthday” is illustrated in Figure 3.

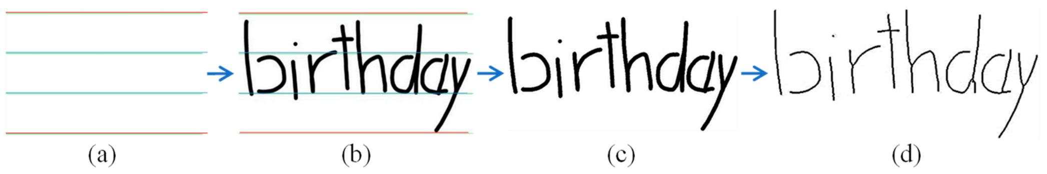


Figure 3. Pre-processing using sample image of “birthday”: (**a**) ruled writing area with four lines, (**b**) input image with handwritten word, (**c**) binarized image, and (**d**) thinned image.

3.1.2. Labeling

After the pre-processing step, objects in the image are labeled in a four-step process. In the first step, connected objects are labeled. Since some characters such as i, j, ? and ! are composed of more than one object, we need to combine labeled objects as the second step. After combining labels, we create a new image with specific spaces between each labeled object to remediate overlapping characters in the third step. Since the projection profile approach is mainly used to segment cursive words, we should connect small disjoints in the fourth step to reduce over-segmented points.

In the proposed system, there are 72 characters (digit, alphabet and special characters). The connected object is referred to a group of connected pixels. Each connected object is assigned with a unique label number. For example, character i has two objects and these two objects need to combine as one label number, which is also called a combined label. In the following Figure 4, label 7 is an example of a left open object. Combining labels change label 7 to label 1 after a left open object is detected.

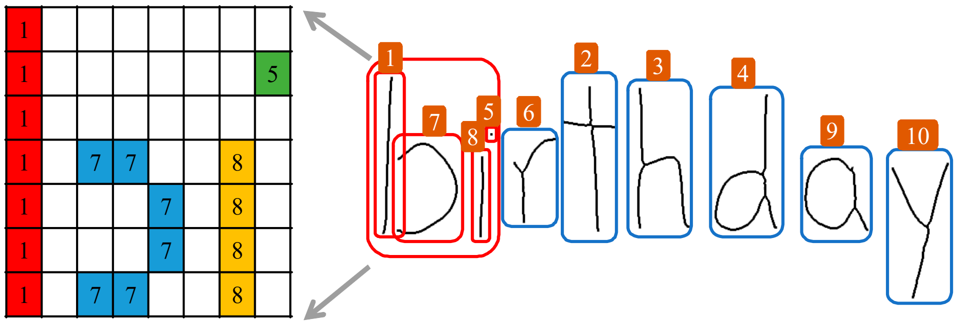


Figure 4. Illustration of connected components labelling with sample table of labels for two characters, bi.

(a) Initial labeling using connected regions:

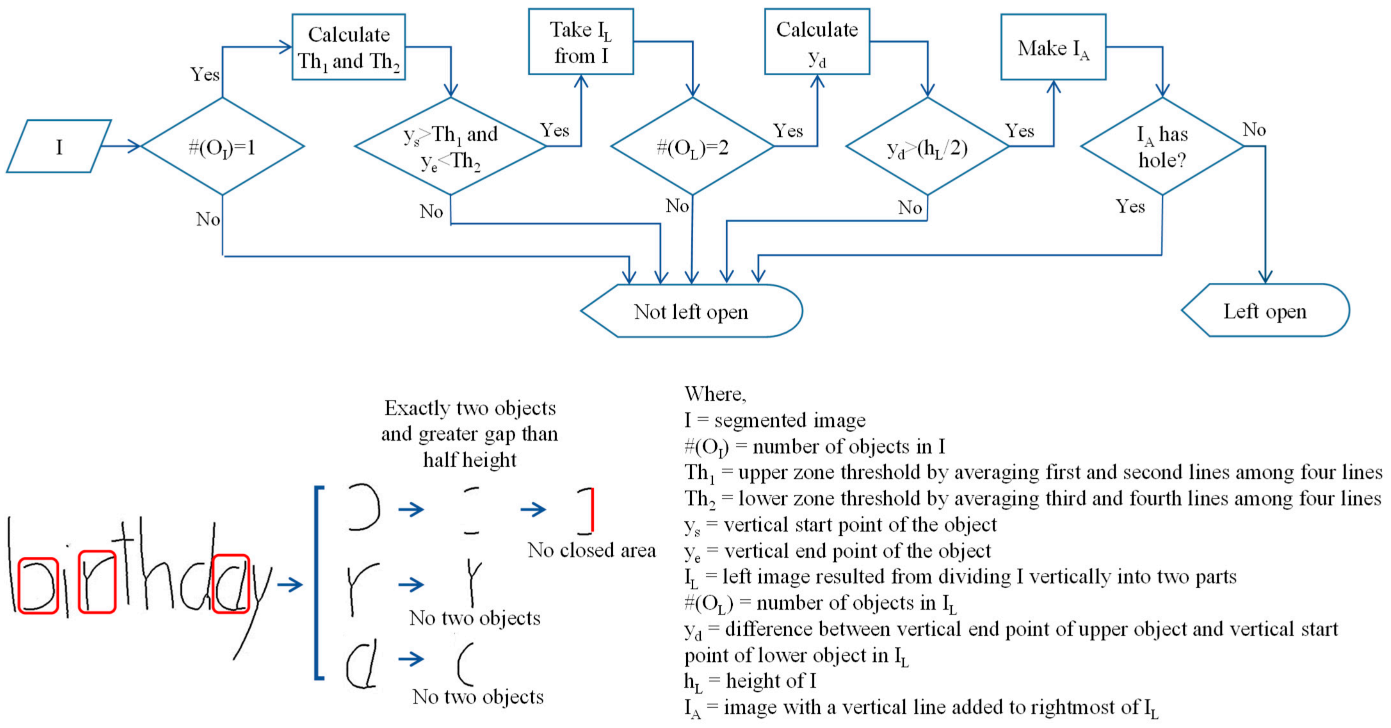
The objects are labeled using the connected component labeling concept. The labeled number, horizontal start and end points, and vertical start and end points are retained for each labeled object. Labels are sorted in ascending order based on the horizontal start point. This allows for analyzing the characters according to the stroke order used in composing them. In the label-combination step, we use horizontal start and end points, as well as vertical start and end points of two labels to extract the location of each label from the image. Next, we replace the larger label with the smaller one. After that, we recalculate connected component objects in the modified data. Then, the label numbers, horizontal start and end points, and vertical start and end points are recalculated, and the labels are resorted to use in the next step. In this way, we correctly label characters.

(b) Label combination for some characters

Generally, we use four categories for objects in the label combination process. They are completely covered objects, partially covered objects, uncovered objects and objects left open. Based on empirical results, we assume that an object with less than a 600-pixel count cannot be a complete character. For combining labels for objects in the first category, if two consecutively written objects are completely covered by each other, and the pixel counts for both objects are less than 600, they are combined. For the second category, if two consecutively written labels are partially covered by each other by at least 20 pixels, they are combined.

For the third category, if objects with pixel counts of less than 600 are found, they are combined with their left or right labels. If the object is the first component of a word, it must be combined with the next object to the right, and if it is last component of a word, it is combined with its left object. If the object is neither the first nor the last component of a word, we must determine whether it is associated with an object to the left or right. To find associated objects, the center points of a found object must be calculated, along with the center points of objects to the left and right. Then, the found object is combined with the closest adjacent object. We should not combine labels if the character is a decimal point. Since the pixel count of a decimal point is also matched with a third constraint, it can be wrongly combined, so we add a constraint for a third category of objects. In this constraint, labels are not combined if the object is horizontally covered by both neighbors.

In the final category, since characters composed of non-contiguous strokes (left open) are not included in the set of recognizable characters in the proposed system, the left-open object is combined with the object to the left. Using the flowchart in Figure 5, we determine whether each labeled object is left open or not. We have illustrated the left-open detection process using the word "birthday" as a sample image. We must detect left-open data for the b character. In the sample image, only three objects (left-open data, r and a) are possibilities after the first two conditions are satisfied. Therefore, we can confirm the existence of left-open data after satisfying the third condition. For detecting characters such as s and g, we consider one final condition, as they might result in two separate objects if divided. However, they cannot have a gap that is greater than half the image height and might contain a hole after the line is added.



Where,
I = segmented image
$\#(O_I)$ = number of objects in I
Th_1 = upper zone threshold by averaging first and second lines among four lines
Th_2 = lower zone threshold by averaging third and fourth lines among four lines
y_s = vertical start point of the object
y_e = vertical end point of the object
I_L = left image resulted from dividing I vertically into two parts
$\#(O_L)$ = number of objects in I_L
y_d = difference between vertical end point of upper object and vertical start point of lower object in I_L
h_L = height of I
I_A = image with a vertical line added to rightmost of I_L

Figure 5. Left open detection flowchart with illustrated image.

(c) Label separation

In the label separation process, we create a new image with the same height as the current image with a new width, as calculated in (1). We use a space pixel count of 20 in our system. We put the first labeled object into a new image, and then add the next object 20 pixels away from the first labeled object's end point.

We place the next object similarly. We place a fixed segment point halfway across the space to the next object:

$$w(I_{new}) = \sum_{i=1}^{n} (x_{ri} - x_{li}) + (n-1) \times k \tag{1}$$

where w (I_{new}) is the width of the new image, x_{ri} is the rightmost x-coordinate of object I, x_{li} is the leftmost x-coordinate of object I, n is the total number of objects, and k is the space pixel count.

(d) Small disjoint connection

In the small disjoint connecting process, we dilate and erode each object label with same structuring element in order to preserve the object size, and to connect small disjoints. If the number of holes in the resulting object is increased by exactly one, we replace the original object with a dilated and eroded object. Otherwise, we retain the original object.

All four steps in the labeling process are illustrated in Figure 6 using the word "birthday".

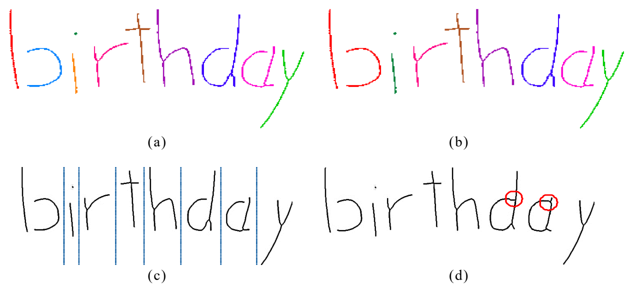


Figure 6. Illustration of four labeling steps: (**a**) Labeling of connected objects, (**b**) combining labels for characters b and i using left open and uncovered label combination categories, (**c**) label separation, and (**d**) small disjoint connection of characters d and a.

3.1.3. Segmentation with Projection

Segmentation in cursive words is performed using vertical projection [26] which is the summation of pixel values in a row for each column. After connecting small disjoints, we perform projection segmentation for each segmented object of the new image. The steps of projection segmentation are described as follows:

Step 1. Find vertical projection values for each x-coordinate in the segmented image using (2) and use the points where projection values are 1:

$$vp(x) = \sum_{y=1}^{m} f(x,y) \tag{2}$$

where $vp(x)$ is the vertical projection or histogram values of the x-coordinate, y is the y-coordinate, x is the x-coordinate, m is the height of the image and $f(x,y)$ is the pixel value at (x,y) of the image.

Step 2. Use an average point if adjacent points are less than 7 pixels apart, in which a character cannot exist in the part segmented with the adjacent points.

Step 3. Retain points that match the following two constraints as projection segment points.

- The first point of the group of adjacent points is not the leftmost point of the partial image.
- The last point of the group of adjacent points is not the rightmost point of the partial image.

This process is illustrated in Figure 7. After performing the above steps for all segments, all projection segment points are combined with the fixed segment points obtained in the label combination step.

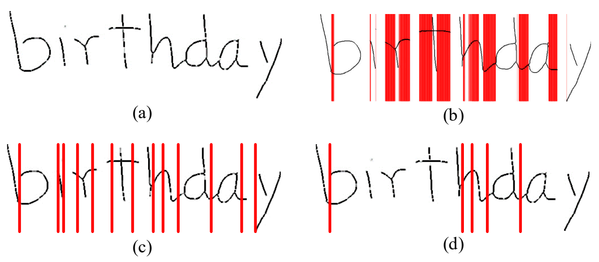


Figure 7. Segmentation based on vertical projection: (**a**) Original image, (**b**) image with segment lines after Step 1, (**c**) image with segment lines after Step 2, and (**d**) image with segment lines after Step 3.

3.1.4. Over-Segments' Removal

Over-segmented points are discarded based on the following three points of logic. These are closed character detection, left open character detection, and pixel count detection [27]. Closed character detection is useful for removing over-segments because an over-segmented character cannot have a loop or a semi-loop in its nearest neighbor so a segment line is discarded if neither of its neighbors has a closed character. A closed character is one with a loop or a semi-loop such as a, c, n and o [28]. A closed character is detected using four pairs of foreground pixel points, as illustrated in Figure 8. In other words, to detect a closed character, we use the following two vertical lines: from a_1 to a_2, and from b_1 to b_2, and we use the following two horizontal lines: from point c_1 to c_2, and d_1 to d_2. In this process, the empirical results indicate that a pixel distance of 20 is best used as the threshold.

1. Find two points, a_1 and a_2, whose x-coordinates are equal, and the differences between y-coordinates are greater than the threshold.
2. Similarly, find an additional two points, b_1 and b_2. In finding these points, the difference between the x-coordinates of a_1 and b_1 must also exceed the threshold.
3. Find c_1 and c_2, in the same way as finding a_1 and a_2, and then find d_1 and d_2 in the same way as finding b_1 and b_2.
4. If none of these four points are zero, the object is determined to be a closed character.

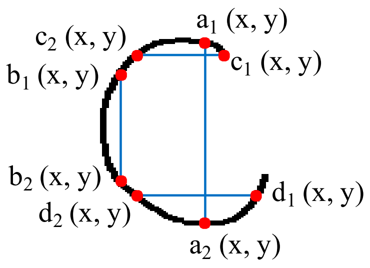


Figure 8. Illustration of closed character determination.

A left open object is again detected using the process described by the flowchart in Figure 5, and an over-segmented point between the object and its neighbor to the left is removed. If the pixel count of an object is less than the threshold (a number that is too small to be a character), we remove the projection segment point for that segment of the object. All three points of logic used in removing over-segmented points are illustrated in Figure 9. In this figure, the blue segmentation lines are drawn using the labeled points. The red lines are drawn using the projection points. The 1st segment line is removed using the left open data constraint, the 6th line is removed using the closed character constraint, and the 7th line is removed using the pixel count constraint.

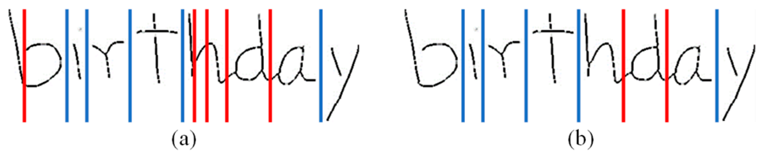


Figure 9. Over-segments removal: (**a**) Image with fixed and projection segment lines, and (**b**) image with correct segment lines.

3.2. Classification

After the handwritten characters are segmented, the final step is to classify the handwritten characters. We applied a convolutional neural network (CNN) model for feature extraction and classification processes. To perform character recognition on mobile phones and Android tablets, we used a framework based on an open-source software library called TensorFlow Lite [29]. We set up the TensorFlow Lite model in our system as described in [30]. TensorFlow was designed and developed by Google Brain. One of the advantages of using TensorFlow Lite is that the learning model does not need to be trained on mobile devices. The learning model is trained on a high-end machine, and then converted into the TensorFlow Lite model file (a .tflite file) by the converter. By using this file, the classification process can be performed on mobile devices.

3.2.1. Image Normalization

In preparing for training and testing, we first performed image size normalization. Each segmented character is normalized into a 64 × 64 pixel width and height. We converted an input image into a binary image, and extracted a region of interest (ROI). Then we performed a resizing operation. To adjust the aspect ratio, we primarily checked the number of rows and columns in the input image. In the first step, we resized the larger side into 64 pixels. In the second step, we padded zero values to the smaller size to fulfill 64 pixels. We inserted zero padding values from left and right for the image so that its width is smaller than its height. When the height of the image is smaller than its width, we inserted zero padding values from top and bottom. While resizing an input image, degradation can occur due to the number of its object pixels. To overcome this problem, we applied a morphological operation called thickening on the image, and then resized it. This process is shown in Figure 10.

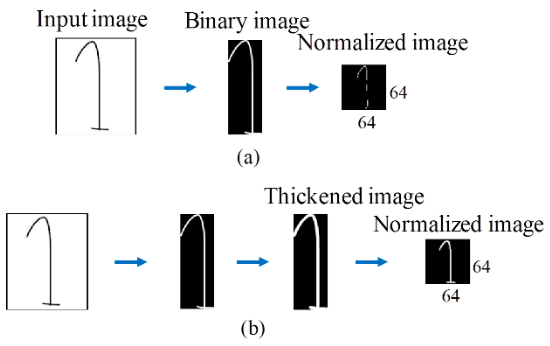


Figure 10. Normalization step on input image: (**a**) Illustration of normalized image without thickening, and (**b**) illustration of normalized image with thickening.

3.2.2. Initial Classification with Convolutional Neural Network

For the CNN model configuration, we applied 20 convolutional filters with the same padding and a filter size of 3 × 3, and then applied a Rectified Linear Unit (ReLU) layer for the first hidden layer. For the second hidden layer, we used 40 filters and a filter size of 5 × 5, as well as a ReLU layer. For a third hidden layer, we used 60 filters with a filter size of 5 × 5 and a ReLU layer, followed by two fully connected layers. The first fully connected layer has 1000 nodes. The size (n) of a second fully connected or classification layer varies in size for each type of classification. In order to downsample the layer size, we performed max pooling after each convolution operation with a size of 2 × 2, and a stride of 2. We proposed different types of classifiers and set the number of nodes for each character type in the final classification layer as follows: digits and special characters (20), uppercase (26), lowercase (26), the combination of uppercase and lowercase (52), digit and lowercase (36) and the combination of all types (72). For the training parameters, we used the Adam optimizer [31] with a learning rate of 0.001 and 4 epochs. The network architecture is illustrated in Figure 11.

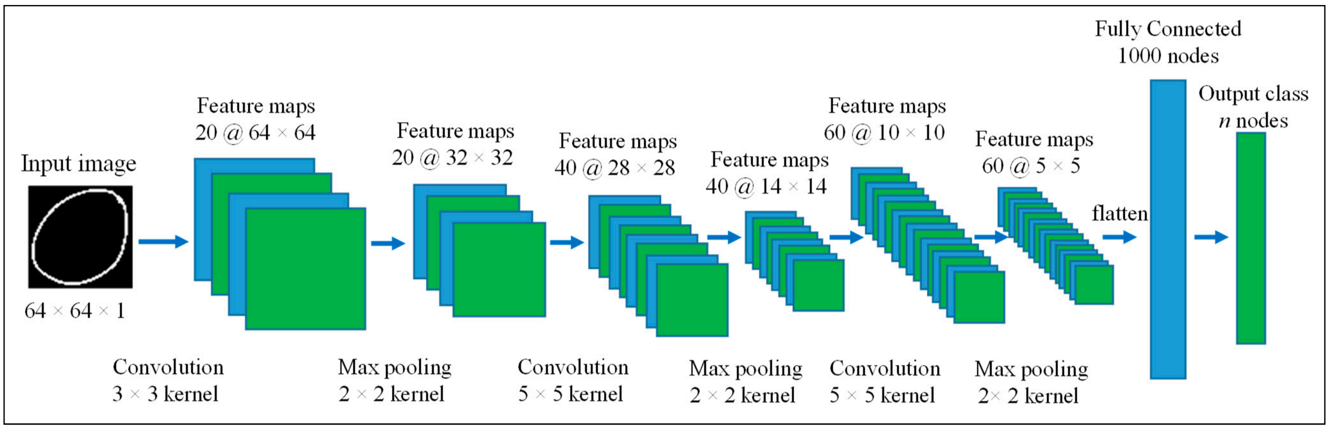


Figure 11. The convolutional neural network (CNN) architecture used in the proposed system.

3.2.3. Differentiating Uppercase and Lowercase Characters

The system is designed to recognize digits, uppercase characters, lowercase characters, and some special characters. In the final result, however, some characters are confused between uppercase and lowercase. In order to solve this problem, we relied on a strength of our proposed system. It provides four lines on a ruled screen as guidelines to help children write. We calculate the threshold value from the first two lines of the four lines area. The midpoint between the first two lines is used as the threshold value. If the start point of the character is under the threshold value, it is considered a lowercase letter and otherwise an uppercase letter. The purpose of differentiating uppercase and lowercase characters is to correctly display on the proposed application. Our list of easily confused characters is the following: C, c, O, o, P, p, S, s, U, u, V, v, W, w, X, x, Y, y, Z and z. If the recognized object is on this list, we used this differentiation approach. This process is illustrated in Figure 12.

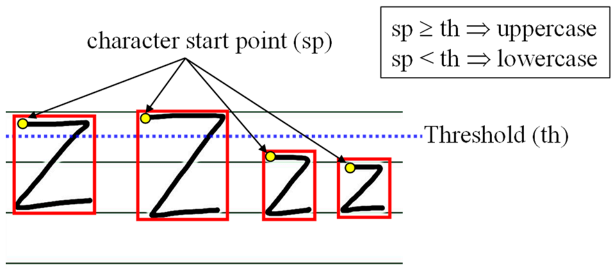


Figure 12. Illustration of uppercase or lowercase determination.

4. Data Collection

All experimental work was performed on a self-collected handwritten dataset. These data were collected from students in the targeted countries, and their grade range was from kindergarten to primary school. Sample rough data are illustrated in Figure 13. We collected handwritten data in the traditional way from Nepal, Myanmar, and the Philippines in the same format. Data from Bangladeshi students were also collected in a traditional way, but by using a different format. We also collected data from students in Myanmar using an Android tablet. We created a simple application to collect data written on an Android tablet. This application features a screen on which to draw, and two buttons: save and clear. The amount of data in our dataset is shown in Table 1.

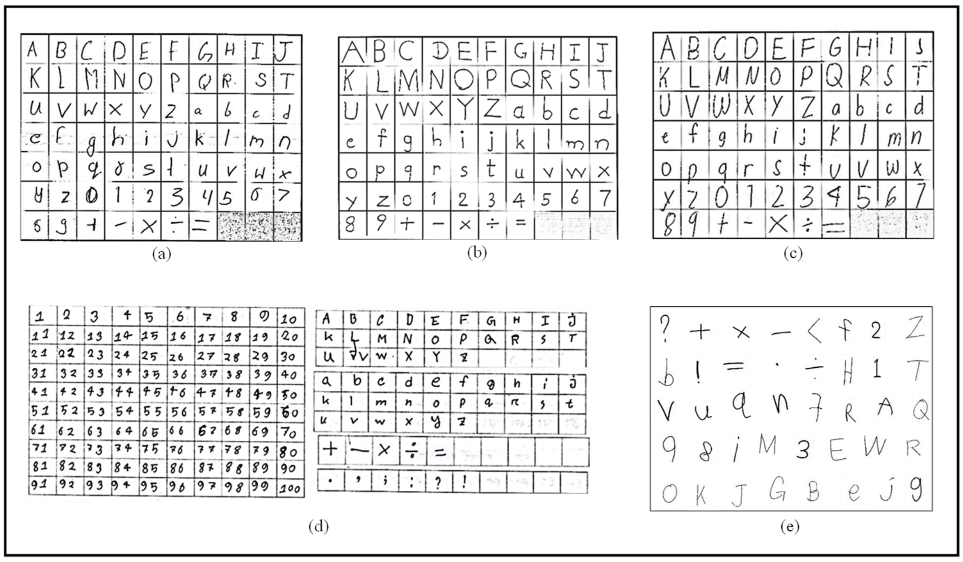


Figure 13. Sample rough data from students in: (**a**) Nepal using the traditional way, (**b**) Myanmar using the traditional way, (**c**) Philippines using the traditional way, (**d**) Bangladesh using the traditional way, and (**e**) Myanmar using an Android tablet.

Table 1. Dataset information.

Dataset Information	Digits	Uppercase	Lowercase	Special Characters	Total
Bangladesh	5107	758	754	350	6969
Nepal	221	600	385	125	1331
Myanmar	451	788	1026	120	2385
Philippines	134	690	628	283	1735
Tablet collected data	2450	6287	6292	2217	17,246
Total	8363	9123	9085	3095	29,666

The proposed system aims to help children write the alphabet, digits, and some mathematical characters. Therefore, we used different types of classifiers for these characters. The classifier types and their respective ID numbers are listed in Table 2. We performed training and testing operations on each classifier. These images were split in an 80:20 ratio between training and testing. The training accuracy for each classifier is also shown in Table 2. While performing the training and testing processes on our self-collected handwritten dataset, the differentiation of uppercase and lowercase condition was not applied. This condition is only applied on Android applications of our proposed system.

Table 2. Training accuracy for each classifier.

No.	Type	ID	Training	
			No. of Images	Accuracy (%)
1	Digits and Special Characters	1	7594	99.25
2	Uppercase	2	6907	98.80
3	Lowercase	3	6749	98.67
4	Uppercase and Lowercase	4	13,656	91.97
5	Digits and Lowercase	6	12,052	95.46
6	All combined	5	21,250	91.63

5. Experimental Results

The proposed CNN model (Mp) is compared with two other CNN models: model 1 (M_1) and model 2 (M_2). Each model has a different architecture, filter size, and output feature map. The detailed architecture for each model is shown in Table 3. In order to downsample the image layer size, we performed max pooling on the ReLU layer after each convolution operation with a size of 2 × 2, and a stride of 2. The first fully connected layers are composed of 80 and 100 nodes for M_1 and M_2. For the second fully connected layer or the classification layer, the output node sizes are same as with the proposed system's architecture. Table 4 provides the testing accuracies of the proposed system, compared with those for models M_1 and M_2. According to the experimental results, the proposed model (Mp) gives higher accuracy than the other compared models.

Table 3. Description of model architectures.

No.	Model Name	Layer	Filter Size	Output Feature Map
1	M1	First Layer	5 × 5	16
		Second Layer	5 × 5	32
		Third Layer	3 × 3	64
2	M2	First Layer	5 × 5	32
		Second Layer	5 × 5	64
		Third Layer	3 × 3	128
3	Mp	First Layer	3 × 3	20
		Second Layer	5 × 5	40
		Third Layer	5 × 5	60

Table 4. Comparison of testing accuracies with different models.

No.	Type	No. of Testing Images	Testing Accuracy (%)		
			M_1	M_2	M_p
1	Digit and Special Characters	3864	95.39	96.84	**98.01**
2	Uppercase	2216	96.34	96.12	**97.43**
3	Lowercase	2336	88.14	89.43	**92.72**
4	Uppercase and Lowercase	4552	80.67	79.13	**82.97**
5	Digit and Lowercase	5396	88.55	88.88	**91.14**
6	All combine	8416	82.32	82.31	**85.74**

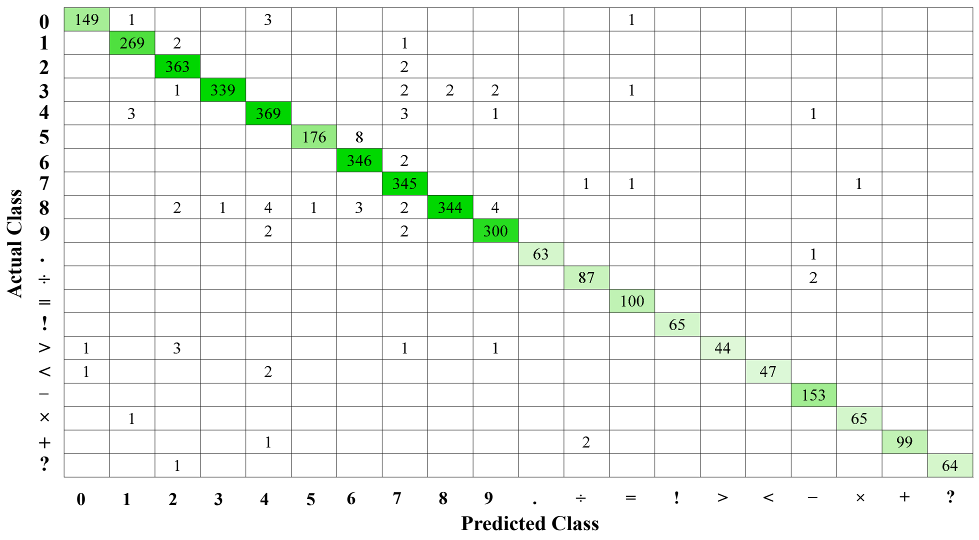
According to our experimental results, the proposed system has a recognition accuracy of over eighty percent for each classification for digits and special characters, uppercase letters, lowercase letters, the combination of uppercase and lowercase letters, digit and lowercase letters, and the combination of all characters.

Our proposed model also makes a comparison with other methods, such as Histogram of Oriented Gradient (HOG) [32] features with the multiclass Support Vector Machine (SVM) classifier and AlexNet [33], on our local handwritten dataset. The same training and testing dataset are applied on both models. For HOG feature extraction, we empirically set a 4 × 4 cell size with 9 bins orientation to extract prominent features from each image. The Alexnet model was set up as described in [34]. For AlexNet implementation on our dataset images, we set a stochastic gradient descent with momentum (SGDM) as the optimizer, a batch size of 64 images and the learning rate to 0.0001 with 4 epochs. The testing accuracies of the proposed model and other compared models are shown in Table 5. According to the experimental result, the proposed system has good recognition accuracy than the other compared models in all classifier types.

Table 5. Comparison of testing accuracies with other compared models.

No.	Type	Testing Accuracy (%)		
		HOG + SVM	AlexNet	M_p
1	Digit and Special Characters	97.05	97.23	**98.01**
2	Uppercase	97.25	97.02	**97.43**
3	Lowercase	90.03	88.36	**92.72**
4	Uppercase and Lowercase	79.70	81.24	**82.97**
5	Digit and Lowercase	88.64	89.60	**91.14**
6	All combine	82.56	84.19	**85.74**

The confusion matrix of the Type 1 classifier (digit and special character) is shown in Figure 14.



Actual Class \ Predicted Class	0	1	2	3	4	5	6	7	8	9	.	÷	=	!	>	<	−	×	+	?
0	149	1			3								1							
1		269	2					1												
2			363					2												
3			1	339				2	2	2			1							
4		3			369			3		1							1			
5						176	8													
6							346	2												
7								345				1	1					1		
8			2	1	4	1	3	2	344	4										
9					2			2		300										
.											63						1			
÷												87					2			
=													100							
!														65						
>	1		3					1		1					44					
<	1				2											47				
−																	153			
×		1																65		
+					1							2							99	
?			1																	64

Figure 14. A confusion matrix of Type 1 classifier by proposed model.

We then developed an Android application for our proposed system. In our application, we used Android studio IDE (version 3.4.1). The OpenCV computer vision library (opencv-3.4.6-android-sdk) is imported into our system to incorporate built-in image processing modules. For the classification process, we used a TensorFlow Lite (version 1.13.1) to transform the trained network file (.h5) into a file (.tflite) that can be applied on Android phones and tablets. The application interface is shown in Figure 15. Students can write in a ruled text area with four lines provided as writing guidelines. This ruled interface is useful in distinguishing similar uppercase and lowercase characters, such as C, c, O, o, P and p. The user can choose between the various types of classification by entering the classifier ID number in classifier type (for example, 1 for digits and special characters). The user can write on the ruled screen with four green guidelines, and then push the classify button to get a classification result.

The classification results for each character and for the number of characters are shown to the user. After adding lecture contents to the application, inputting the classifier type will no longer be necessary.

We tested our proposed system using 1000 randomly selected words written by children. Table 6 provides the segmentation accuracy by word. The recognition accuracy is calculated in two ways: for words and for characters. The results are described in Table 7. Some correctly recognized words together with their segmentation lines and experimental results are shown in Figure 16.

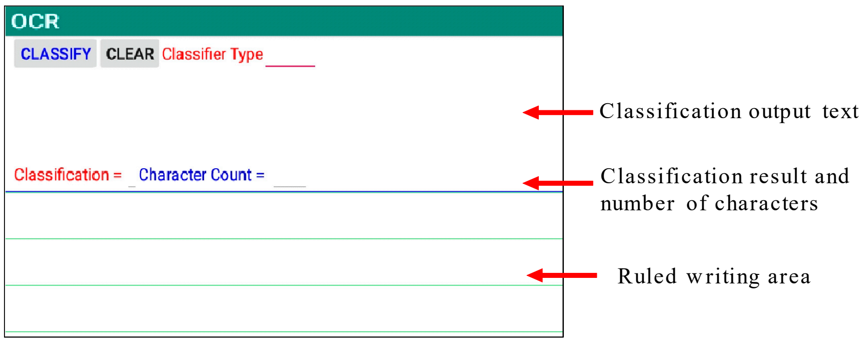


Figure 15. A simple user interface for the system.

Table 6. Segmentation results.

Result	Correct Count	Incorrect Count	Total Count	Accuracy (%)
Words	985	15	1000	98.50

Table 7. Recognition results.

Result	Correct Count	Incorrect Count	Total Count	Accuracy (%)
Words	956	44	1000	95.60
Characters	4550	57	4607	98.76

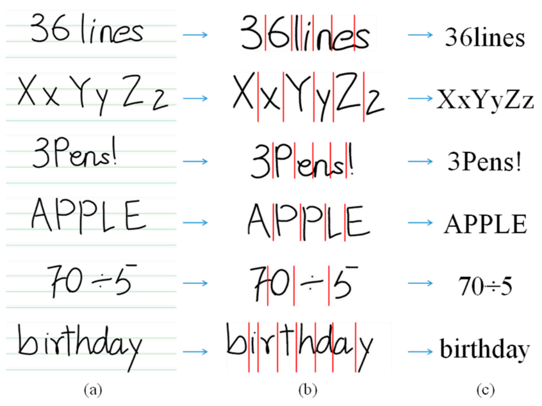


Figure 16. Some images of correct segmentation and recognition: (**a**) Images input to the system, (**b**) images with segmentation lines, and (**c**) recognition results.

6. Limitations

Based on these results, our system cannot correctly segment for some characters such as a disjoint U or d. This is because U and LI are easily confused, as are d and cl. Wrongly segmented and recognized words are shown in Figures 17 and 18, respectively. The recognition results for some messy images are shown in Figure 19. The overall accuracy of our proposed system on the 1000 words in our randomly tested data was 95.60%, and the overall accuracy on 4607 characters was 98.76%.

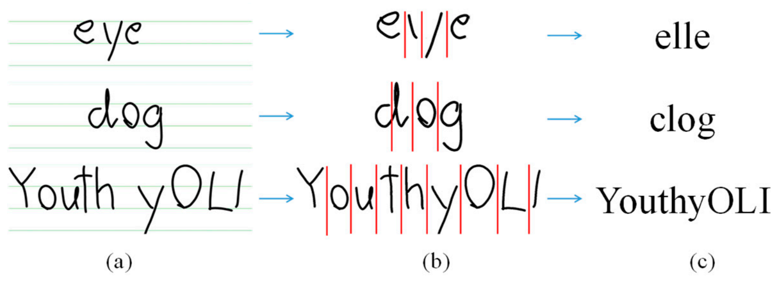


Figure 17. Some images of correct segmentation and recognition: (**a**) Images input into the system, (**b**) images with segmentation lines, and (**c**) recognition results.

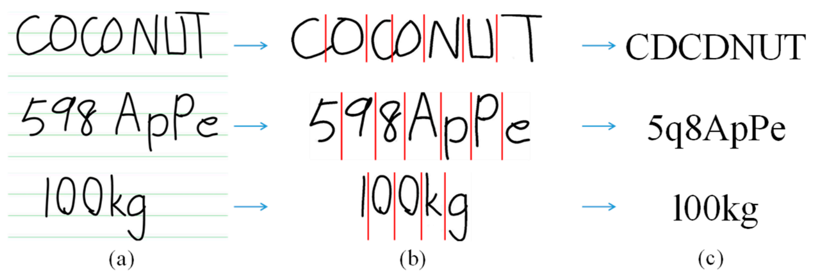


Figure 18. Some images of incorrect recognition results: (**a**) Images input into the system, (**b**) images with segmentation lines, and (**c**) recognition results.

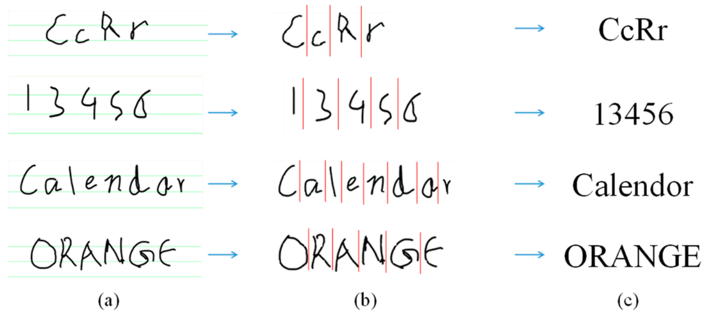


Figure 19. Recognition results for some messy images: (**a**) Images input into the system, (**b**) images with segmentation lines, and (**c**) recognition results.

7. Conclusions

In this paper, we proposed an application for handwritten characters that can be applied as a teaching aid for children. The application enables self-study, so that children can learn without experienced teachers. We also proposed a character segmentation processes to overcome the segmentation problems. For character recognition, we applied the CNN classifier and processed each segmented character. The recognition time for one character took approximately 0.3 s on a mobile phone or tablet, and it is an acceptable time. In future work, we will combine the word-checking process in our current system to check answers immediately. Moreover, we will collect more handwritten data to cover the various handwriting styles.

Author Contributions: Conceptualization, T.T.Z.; methodology, T.T.Z., S.T. and M.Z.P.; software, S.T. and M.Z.P.; validation, T.T.Z., S.T. and M.Z.P.; resources, T.T.Z. and T.O.; writing—original draft preparation, T.T.Z., S.T. and M.Z.P.; writing—review and editing, T.T.Z., S.T. and M.Z.P.; visualization, S.T. and M.Z.P.; supervision, T.T.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported in part by the Strategic Foundational Technology Improvement Support Operation, Japan.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Sudarsana, K.; Nakayanti, A.R.; Sapta, A.; Haimah; Satria, E.; Saddhono, K.; Daengs GS, A.; Putut, E.; Helda, T.; Mursalin, M. Technology Application in Education and Learning Process. *J. Phys. Conf. Ser.* **2019**, *1363*, 1–6. [CrossRef]
2. Education: Out-of-School Rate for Children of Primary School Age. 2020. Available online: https://data.uis.unesco.org/index.aspx?queryid=123 (accessed on 20 July 2020).
3. Weinstein, J. The Problem of Rural Education in the Philippines. 2010. Available online: https://developeconomies.com/development-economics/the-problem-of-education-in-the-philippines/ (accessed on 20 July 2020).
4. Butler, C.; de Pimenta, R.; Fuchs, C.; Tommerdahi, J.; Tamplain, P. Using a handwriting app leads to improvement in manual dexterity in kindergarten children. *Res. Learn. Technol.* **2019**, *27*, 1–10. [CrossRef]
5. Pegrum, M.; Oakley, G.; Faulkner, R. Schools going mobile: A study of the adoption of mobile handheld technologies in Western Australian independent schools. *Australas. J. Educ. Technol.* **2013**, *29*, 66–81. [CrossRef]
6. Stephanie, "Handwriting Apps for Kids". Available online: https://parentingchaos.com/handwriting-apps-for-kids/ (accessed on 20 July 2020).
7. Baldominos, A.; Saez, Y.; Isasi, P. A survey of handwritten character recognition with mnist and emnist. *Appl. Sci.* **2019**, *9*, 3169. [CrossRef]
8. Dewa, C.K.; Fadhilah, A.L.; Afiahayati, A. Convolutional neural networks for handwritten Javanese character recognition. *Indones. J. Comput. Cybern. Syst.* **2018**, *12*, 83–94. [CrossRef]
9. Lincy, R.B.; Gayathri, R. Optimally configured convolutional neural network for Tamil Handwritten Character Recognition by improved lion optimization model. *Multimed. Tools Appl.* **2021**, *80*, 5917–5943. [CrossRef]
10. Zhang, Z.; Tang, Z.; Wang, Y.; Zhang, Z.; Zhan, C.; Zha, Z.; Wang, M. Dense Residual Network: Enhancing global dense feature flow for character recognition. *Neural Netw.* **2021**, *139*, 77–85. [CrossRef]
11. Asif, A.M.A.M.; Hannan, S.A.; Perwej, Y.; Vithalrao, M.A. An Overview and Applications of Optical Character Recognition. *Int. J. Adv. Res. Sci. Eng.* **2014**, *3*, 261–274.
12. Zin, T.T.; Misawa, S.; Pwint, M.Z.; Thant, S.; Seint, P.T.; Sumi, K.; Yoshida, K. Cow Identification System using Ear Tag Recognition. In Proceedings of the 2020 IEEE 2nd Global Conference on Life Sciences and Technologies (LifeTech), Kyoto, Japan, 10–12 March 2020; pp. 65–66.
13. Zin, T.T.; Pwint, M.Z.; Seint, P.T.; Thant, S.; Misawa, S.; Sumi, K.; Yoshida, K. Automatic Cow Location Tracking System using Ear Tag Visual Analysis. *Sensors* **2020**, *20*, 3564. [CrossRef]
14. Manjunath, A.E.; Sharath, B. Implementing Kannada Optical Character Recognition on the Android Operating System for Kannada Sign Boards. *Int. J. Adv. Res. Comput. Commun. Eng.* **2013**, *2*, 932–937.
15. Perwej, Y.; Chaturvedi, A. Neural Networks for Handwritten English Alphabet Recognition. *Int. J. Comput. Appl.* **2011**, *20*, 1–5. [CrossRef]
16. Gosavi, M.B.; Pund, I.V.; Jadhav, H.V.; Gedam, S.R. Mobile Application with Optical Character Recognition using Neural Network. *Int. J. Comput. Sci. Mob. Comput.* **2015**, *4*, 483–489.
17. Anil, R.; Manjusha, K.; Kumar, S.S.; Soman, K.P. Convolutional Neural Networks for the Recognition of Malayalam Characters. *Adv. Intell. Syst. Comput.* **2015**, *328*, 493–500. [CrossRef]
18. Ly, N.T.; Nguyen, C.T.; Nguyen, K.C.; Nakagawa, M. Deep convolutional recurrent network for segmentation-free offline handwritten Japanese text recognition. In Proceedings of the 14th IAPR International Conference on Document Analysis and Recognition (ICDAR), Kyoto, Japan, 9–15 November 2017; IEEE: Piscataway, NJ, USA, 2017; Volume 7, pp. 5–9.
19. Xue, W.; Li, Q.; Xue, Q. Text detection and recognition for images of medical laboratory reports with a deep learning approach. *IEEE Access* **2019**, *8*, 407–416. [CrossRef]
20. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Trans. Pattern Anal. Mach. Intell.* **2016**, *39*, 1137–1149. [CrossRef] [PubMed]
21. Shorim, N.; Ghanim, T.; AbdelRaouf, A. Implementing Arabic Handwritten Recognition Approach using Cloud Computing and Google APIs on a mobile application. In Proceedings of the 14th International Conference on Computer Engineering and Systems (ICCES), Cairo, Egypt, 17 December 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 88–95.
22. Ghanim, T.M.; Khalil, M.I.; Abbas, H.M. Multi-stage offline arabic handwritin recognition approach using advanced cascading technique. In Proceedings of the 8th International Conference on Pattern Recognition Applications and Methods (ICPRAM), Prague, Czech Republic, 19–21 February 2019; pp. 532–539.
23. Zin, T.T.; Thant, S.; Htet, Y. Handwritten Characters Segmentation using Projection Approach. In Proceedings of the 2020 IEEE 2nd Global Conference on Life Sciences and Technologies (LifeTech), Kyoto, Japan, 10–12 March 2020; pp. 107–108.
24. Vaidya, R.; Trivedi, D.; Satra, S.; Pimpale, M. Handwritten Character Recognition Using Deep-Learning. In Proceedings of the Second International Conference on Inventive Communication and Computational Technologies (ICICCT), Coimbatore, India, 20–21 April 2018; pp. 772–775. [CrossRef]
25. Weng, Y.; Xia, C. A new deep learning-based handwritten character recognition system on mobile computing devices. *Mob. Netw. Appl.* **2019**, *25*, 402–411. [CrossRef]
26. Anupama, N.; Rupa, C.; Reddy, E.S. Character Segmentation for Telugu Image Document using Multiple Histogram Projections. *Glob. J. Comput. Sci. Technol.* **2013**, *13*, 11–15.

27. Brodowska, M. Oversegmentation Methods for Character Segmentation in Off-line Cursive Handwritten Word Recognition: An Overview. *Schedae Inform.* **2011**, *20*, 43–65.
28. Saba, T.; Rehman, A.; Sulong, G. Cursive Script Segmentation with Neural Confidence. *Int. J. Comput. Inf. Control* **2011**, *7*, 4955–4964.
29. TensorFlow: For Mobile & IoT. Available online: https://www.tensorflow.org/lite (accessed on 20 July 2020).
30. Stanek, M. Mobile Intelligence—TensorFlow Lite Classification on Android (added support for TF2.0). 2019. Available online: https://proandroiddev.com/mobile-intelligence-tensorflow-lite-classification-on-android-c081278d9961 (accessed on 20 July 2020).
31. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2015**, arXiv:1412.6980.
32. Dalal, N.; Triggs, B. Histograms of oriented gradients for human detection. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), San Diego, CA, USA, 20–25 June 2005; IEEE: Piscataway, NJ, USA, 2005; Volume 1, pp. 886–893.
33. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. *Commun. ACM* **2017**, *60*, 84–90. [CrossRef]
34. MathWorks, "Transfer Learning Using AlexNet". Available online: https://www.mathworks.com/help/deeplearning/ug/transfer-learning-using-alexnet.html (accessed on 1 March 2021).