Hindawi
Applied Computational Intelligence and Soft Computing
Volume 2021, Article ID 1682395, 14 pages
https://doi.org/10.1155/2021/1682395

Research Article

Proposing Algorithm Using YOLOV4 and VGG-16 for Smart-Education

Phat Nguyen Huu and Khang Doan Xuan

School of Electrical and Electronic Engineering, Hanoi University of Science and Technology, Hanoi, Vietnam

Correspondence should be addressed to Phat Nguyen Huu; phat.nguyenhuu@hust.edu.vn

Received 19 September 2021; Accepted 10 November 2021; Published 29 December 2021

Academic Editor: Ridha Ejbali

In this paper, we propose an algorithm to identify and solve systems of high-order equations. We rely on traditional solution methods to build algorithms to solve automated equations based on deep learning. The proposal method includes two main steps. In the first step, we use YOLOV4 (Kumar et al. 2020; Canu, 2020) to recognize equations and letters associated with the VGG-16 network (Simonyan and Zisserman, 2015) to classify them. We then used the SymPy model to solve the equations in the second step. Data are images of systems of equations that are typed and designed by ourselves or handwritten from other sources. Besides, we also built a web-based application that helps users select an image from their devices. The results show that the proposed algorithm is set out with 95% accuracy for smart-education applications.

1. Introduction

Today, artificial intelligence (AI) has created a revolution in society with the remarkable development of science and technology, especially since their explosion. The applications of AI are applied to many aspects of life that make human life more and more comfortable. There are many applications of AI and deep learning, for example, Google translate tool, facial recognition system, cancer detection through X-ray pictures, self-driving cars of Tesla, and smart-education applications. These are very practical applications for life.

Solving equations is a fundamental problem involving many different applications for learning and researching process. We will solve not only primary equations but also higher equations with complex forms. As a result, it is difficult to consider what the type of problem is and how it will be solved.

Manual computation with simple equations such as hidden cubic, quadratic, and first-degree equations or systems of first-order equations is usually not too difficult. However, it is difficult to find a solution when encountering many unknown parameters of systems of equations. In that case, we have to try many times with different values. After having a solution, we will perform the next steps. However, testing of values is difficult to perform by hand. Therefore, we completely solve this difficult problem specifically by computer vision with the current development of deep learning (DL).

The goal of the paper is to identify and solve an equation or system of high-order equations. Our proposed algorithm has four new points, as follows.

Firstly, we rely on traditional solution methods to build algorithms to solve automated equations using DL.

Secondly, the authors [1] use the YOLOv4 model to optimize the speed and accuracy of objects while detecting them. The results show that the model is suitable for detecting objects at a real-time speed of 65 frames per second (FPS) on the Tesla V100, although accuracy is not good with 43.5% AP and 65.7% AP50 for the MS COCO dataset. The authors [2] use simple online and real-time tracking (SORT) to track objects for long periods of occlusions that will reduce the number of identity switches. The algorithm is performed at 20 Hz and ID switches are reduced from 1423 to 781 (45%). Therefore, we use YOLOv4, an algorithm that has been evaluated to be highly feasible [3, 4], to identify equations in the identification block.

Thirdly, we can see that the accuracy of VGG-16 is similar to that of VGG-19 while the number of parameters is smaller, as shown in Table 1. Therefore, we use VGG-16 [5] to increase accuracy for character recognition block for final block.

Finally, we built datasets by ourselves from different sources.

The rest of the paper is presented as follows. In Section 2, we present related work. In Sections 3 and 4, we present and evaluate the effectiveness of the proposed model, respectively. Finally, we give conclusion in Section 5.

Table 1: Evaluating parameters of CNN based on ImageNet dataset from Keras [5].

Network	Size of network (MB)	Accuracy	Parameter
VGG-16	**528**	**0.9001**	**138,357,544**
VGG-19	549	0.9000	143,667,240

VGG stands for the visual geometry group that has architecture with multiple layers. The number of layers with VGG-16 and VGG-19 is 16 and 19, respectively. VGG-16 data are shown in bold.

2. Related Work

Identifying and solving problems about equations and systems of high-order equations is one of many practical applications of DL [6–20]. The solutions are deployed on many applications such as Quanda, Photomath, and Mathway with millions of users on iOS and Android.

Many solutions are given based on handwriting recognition [6, 7]. A diagram to recognize mathematical expressions based on SVM and simple mathematical expressions is proposed in [8]. The paper focuses on many techniques used for extraction and identification features. The authors [21] transformed technical and scientific literature into electronic form by memorizing mathematical symbols and expressions. Features are calculated using image center and bounding box techniques that have been suggested. Classification is performed by a neural network with an accuracy up to 90%. The authors [22] have proposed a diagonal feature extraction technique for handwritten characters using a feed-forward neural network algorithm. The technique uses vertical, horizontal, and diagonal features for character classification.

Recognition of mathematical symbols using convolutional neural network (CNN) is also proposed [9, 10, 13, 16, 18, 20]. In recent years, CNN has produced a series of research results in image classification arrays. An offshoot of AI neural networks called DL has shown good potential in solving taxonomy problems. DL began to evolve in 2012 when Alex et al. demonstrated their network architecture called AlexNet [10] for image classification to produce outstanding results.

In [20], the authors proposed to extract characters by the equation segmentation method and character by image processing technique. The extracted characters will be put into the CNN model. The sorted string will be processed for solving and give better results. The authors [23] propose a system of model equations for solving a wide range of verbal algebra problems. The authors [24] try to handle arithmetic problems with multiple steps and operations that do not depend on annotations or predefined patterns. The authors [25] present a new way of learning how to use formulas to solve simple addition problems in arithmetic.

In [18], the authors propose a method for solving algebraic to identify parameters in heat conduction equations with classic Tikhonov regularization algorithms. The results show that the regularization method based on the dynamic system method is more effective than Tikhonov. In [13], the authors propose a method of mathematical formula identification for PDF documents. The results show that the proposal method improves accuracy by up to 80%. In [16], the authors propose prealgebraic problem solving of fifth-year students before and after applying the math model. The results show that the method is able to solve many challenging arithmetic word problems. In [17], the authors proposed a regularization strategy-mollification method to analyze the stability of a problem. The results show that the proposed method is effective and stable.

In [20], the authors perform recognition and solution for quadratic equation that is written by hand. The authors use the NIST dataset similar to the famous MNIST dataset for digital characters with each character consisting of 2000 images for data training. They proposed line and character segmentation to separate them from the image for classification. Since characters are extracted from images using line segmentation and character segmentation which are two purely image processing methods, handwriting characters may stick together or italic images. Besides, the scope of problem is confined to square.

Based on the abovementioned analysis results, we propose to identify equations and letters by combining YOLOV4 and VGG-16. The scope of our problems also extends to solving equations or systems of high-order equations that are typed, handwritten, or from other sources. To the best of our knowledge, using CNN for solving mathematical equations has not been considered in the literature.

3. Proposal Solution

3.1. Overview System. Proposal system is described in Figure 1.

In the first step, we will identify equations with model training by YOLOv4 and a system of image datasets that are gathered from multiple sources. In the step, if we encounter skewed images, their bounding boxes will be partially affected by the equations above or below. When an image is tilted, characters are also skewed. Therefore, recognizing and classifying each character will not be accurate.

To solve this problem, we will rotate the image based on the bounding boxes that were identified in the first step. After rotating the image, we will detect the equation again. We will then continue to use YOLOv4 to recognize characters and use the VGG-16 model to classify them based on bounding boxes. Finally, we will have a raw string. We need to process them to put them into the SymPy module. SymPy will immediately give the result. To make system interoperable, we will deploy the model on framework calling Flask.

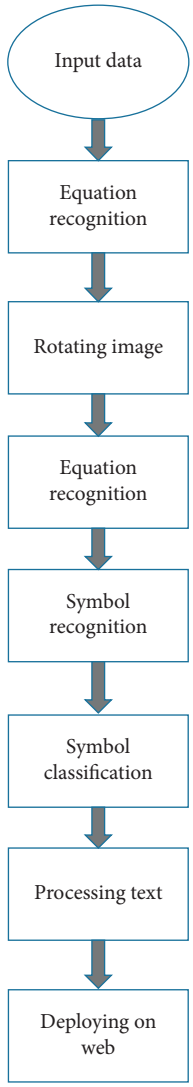


Figure 1: Diagram of the proposal system.

3.2. Implementation Steps

3.2.1. Equation Recognition. YOLO is the best one while comparing the methods using YOLO [26], single shot detector (SSD) [27], R-CNN [28], and faster R-CNN [29]. This algorithm involves CNN (the original version GoogLeNet is called Darknet) that divides the input into grids and cells. Each cell predicts directly bounding box and classifies object. It is highly capable in object recognition and is compact and capable of running in real time at high speed while deploying on devices. Therefore, we will use YOLOV4 for recognizing the equation [3, 4].

Implementation methods include four steps as follows:

(i) Step 1: preparing data. Firstly, we need to prepare data from online sources (such as Google and Internet) or data by ourselves or data from textbooks as shown in Figure 2. In this case, the number of images to train with Yolov4 is 1046.

(ii) Step 2: labeling the data. In this step, we define the bounding boxes of equations based on manual labeling. In the paper, we will use the LabelImg tool. This process is essentially drawing boxes around equations of image. Figure 3 shows an example using the LabelImg tool that automatically generates an "a.txt" file. The file describes the position of the equation of the image.

(iii) Step 3: running model on Google Colab. To train our dataset, we will use a server powered by Google Colab. It is possible to run Python code and libraries to take full advantage of GPUs.

There are several implementations of the YOLO algorithm. In this paper, we use the Darknet algorithm. It is an open source writing by C and CUDA that serves as the foundation of YOLO. It has the advantage of been supported by both the CPU and GPU at same time. The Darknet is used as a framework for YOLO training. We proceed to upload the folder containing the images and the corresponding.txt file that is being labeled after downloading to Google Drive.

In the next step, we will proceed to edit the "yolov4.cfg" file in the Darknet folder as follows:

(1) Changing line subdivisions to 64
(2) Changing classes from 80 to 1 at lines 610, 696, and 783
(3) Changing valuable filters from 255 to 24 at lines 603, 689, and 776
(4) Changing batch stream to 64
(5) Setting max_batches to $n \times 2000$ where n is number of selecting classes equal to
(6) Setting width and height to a multiple of 32 where we set as 416×416

As a final step, we will download pretrained weights of "yolov4.conv.137" from AlexeyAB and perform training.

When we have a data with a ".txt" file generated by LabelImg, we will upload it to Google Colab to train. The results are shown in Figure 4.

(iv) Step 4: identifying equation with the training model. We get weight contained in the "yolov4_-training_2000.weights" file after training in the 2000 epoch. We will use weight and configure parameters for the OpenCV library to identify equations.

We will perform equation recognition with a training model after training data. The results are shown in Figure 5.

3.2.2. Image Rotation. Since all of our images are not taken straight, recognition is inaccurate. Therefore, we will rotate them since equations or systems of equations are recognized more accurately [30–32].

The steps for implementation are as follows:

(i) Step 1: identifying equation and its area. The recognition equation and its area include two small steps. Firstly, we need to identify the area that

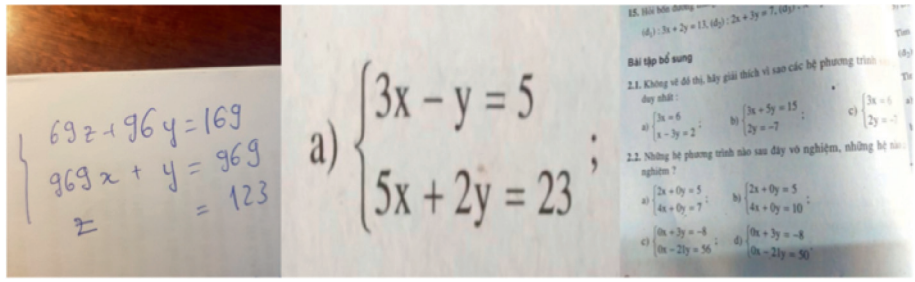


Figure 2: Preparing your data.

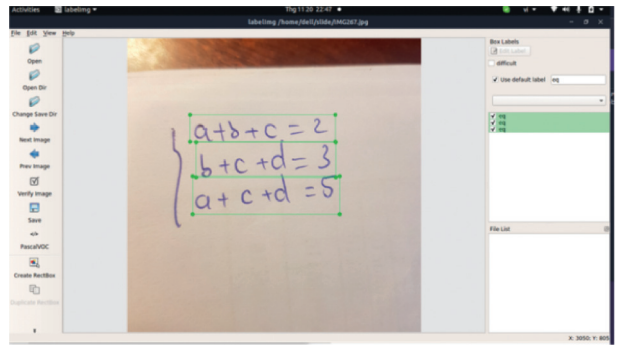


Figure 3: An example of labeling data.

```
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 139 Avg (IOU: 0.902216, GIOU: 0.899895), Class: 0.998279, Obj: 0.802591, No Obj: 0.000059, .5R: 1.000000, .75R: 1.000000, count: 2, class_loss = 0.026073,
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 150 Avg (IOU: 0.839647, GIOU: 0.837219), Class: 0.999505, Obj: 0.788407, No Obj: 0.006660, .5R: 1.000000, .75R: 0.891892, count: 37, class_loss = 1.758231,
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 161 Avg (IOU: 0.860267, GIOU: 0.857394), Class: 0.999064, Obj: 0.931188, No Obj: 0.016539, .5R: 1.000000, .75R: 0.970588, count: 34, class_loss = 0.228717,
 total_bbox = 4964402, rewritten_bbox = 0.072959 %
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 139 Avg (IOU: 0.000000, GIOU: 0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.000005, .5R: 0.000000, .75R: 0.000000, count: 1, class_loss = 0.000125,
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 150 Avg (IOU: 0.858410, GIOU: 0.855477), Class: 0.999560, Obj: 0.886225, No Obj: 0.006546, .5R: 1.000000, .75R: 0.970588, count: 34, class_loss = 0.436196,
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 161 Avg (IOU: 0.837002, GIOU: 0.833985), Class: 0.997602, Obj: 0.948938, No Obj: 0.013177, .5R: 1.000000, .75R: 0.880000, count: 25, class_loss = 0.324276,
 total_bbox = 4964461, rewritten_bbox = 0.072959 %
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 139 Avg (IOU: 0.000000, GIOU: 0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.000060, .5R: 0.000000, .75R: 0.000000, count: 1, class_loss = 0.033311,
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 150 Avg (IOU: 0.819754, GIOU: 0.814405), Class: 0.999280, Obj: 0.806161, No Obj: 0.007995, .5R: 0.978261, .75R: 0.847826, count: 46, class_loss = 1.905951,
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 161 Avg (IOU: 0.830496, GIOU: 0.825435), Class: 0.998542, Obj: 0.910739, No Obj: 0.018316, .5R: 1.000000, .75R: 0.838710, count: 31, class_loss = 0.822549,
 total_bbox = 4964538, rewritten_bbox = 0.072957 %
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 139 Avg (IOU: 0.811675, GIOU: 0.805614), Class: 0.998942, Obj: 0.729665, No Obj: 0.000354, .5R: 1.000000, .75R: 0.750000, count: 12, class_loss = 0.841685,
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 150 Avg (IOU: 0.854422, GIOU: 0.850867), Class: 0.999071, Obj: 0.894714, No Obj: 0.009970, .5R: 1.000000, .75R: 0.884615, count: 78, class_loss = 2.369695,
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 161 Avg (IOU: 0.871611, GIOU: 0.867915), Class: 0.999629, Obj: 0.950344, No Obj: 0.011518, .5R: 1.000000, .75R: 1.000000, count: 26, class_loss = 0.100101,
 total_bbox = 4964654, rewritten_bbox = 0.072956 %
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 139 Avg (IOU: 0.783533, GIOU: 0.775530), Class: 0.998674, Obj: 0.341635, No Obj: 0.000150, .5R: 1.000000, .75R: 0.600000, count: 5, class_loss = 0.644453,
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 150 Avg (IOU: 0.820700, GIOU: 0.816521), Class: 0.999370, Obj: 0.761050, No Obj: 0.012546, .5R: 0.976471, .75R: 0.847059, count: 85, class_loss = 7.088885,
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 161 Avg (IOU: 0.811388, GIOU: 0.806979), Class: 0.997373, Obj: 0.771463, No Obj: 0.013763, .5R: 1.000000, .75R: 0.774194, count: 31, class_loss = 2.068154,
 total_bbox = 4964775, rewritten_bbox = 0.072954 %
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 139 Avg (IOU: 0.840555, GIOU: 0.832094), Class: 0.997818, Obj: 0.468521, No Obj: 0.000788, .5R: 0.966667, .75R: 0.933333, count: 30, class_loss = 3.319150,
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 150 Avg (IOU: 0.830643, GIOU: 0.825681), Class: 0.999186, Obj: 0.690866, No Obj: 0.021453, .5R: 0.989418, .75R: 0.888889, count: 189, class_loss = 15.00754
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 161 Avg (IOU: 0.835758, GIOU: 0.829717), Class: 0.999243, Obj: 0.760867, No Obj: 0.024911, .5R: 0.964912, .75R: 0.912281, count: 57, class_loss = 4.341112,
 total_bbox = 4965051, rewritten_bbox = 0.072970 %
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 139 Avg (IOU: 0.755883, GIOU: 0.730023), Class: 0.997841, Obj: 0.439689, No Obj: 0.000779, .5R: 0.896552, .75R: 0.689655, count: 29, class_loss = 4.560573,
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 150 Avg (IOU: 0.807879, GIOU: 0.801396), Class: 0.999184, Obj: 0.750821, No Obj: 0.012766, .5R: 0.951923, .75R: 0.769231, count: 104, class_loss = 5.912441
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 161 Avg (IOU: 0.850824, GIOU: 0.847627), Class: 0.998796, Obj: 0.796210, No Obj: 0.018741, .5R: 1.000000, .75R: 0.871795, count: 39, class_loss = 1.979034,
 total_bbox = 4965223, rewritten_bbox = 0.072968 %
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 139 Avg (IOU: 0.763086, GIOU: 0.758226), Class: 0.998289, Obj: 0.257239, No Obj: 0.000539, .5R: 0.952381, .75R: 0.619048, count: 21, class_loss = 3.944601,
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 150 Avg (IOU: 0.812389, GIOU: 0.807042), Class: 0.999108, Obj: 0.671565, No Obj: 0.016891, .5R: 0.992701, .75R: 0.737226, count: 137, class_loss = 12.72431
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 161 Avg (IOU: 0.837136, GIOU: 0.833858), Class: 0.999181, Obj: 0.754598, No Obj: 0.026473, .5R: 1.000000, .75R: 0.865672, count: 67, class_loss = 3.535925,
 total_bbox = 4965448, rewritten_bbox = 0.072964 %
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 139 Avg (IOU: 0.776719, GIOU: 0.769768), Class: 0.997861, Obj: 0.204645, No Obj: 0.000790, .5R: 0.971429, .75R: 0.657143, count: 35, class_loss = 7.393345,
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 150 Avg (IOU: 0.833399, GIOU: 0.830515), Class: 0.999136, Obj: 0.646836, No Obj: 0.018908, .5R: 1.000000, .75R: 0.818182, count: 165, class_loss = 14.39657
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 161 Avg (IOU: 0.848365, GIOU: 0.844561), Class: 0.999187, Obj: 0.765276, No Obj: 0.026518, .5R: 1.000000, .75R: 0.846154, count: 65, class_loss = 3.814234,
 total_bbox = 4965713, rewritten_bbox = 0.072960 %
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 139 Avg (IOU: 0.814954, GIOU: 0.811969), Class: 0.998416, Obj: 0.372532, No Obj: 0.000705, .5R: 0.964286, .75R: 0.785714, count: 28, class_loss = 3.823346,
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 150 Avg (IOU: 0.817963, GIOU: 0.814932), Class: 0.999086, Obj: 0.700890, No Obj: 0.017369, .5R: 0.993865, .75R: 0.785276, count: 163, class_loss = 11.54073
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 161 Avg (IOU: 0.826407, GIOU: 0.822361), Class: 0.998137, Obj: 0.804245, No Obj: 0.025727, .5R: 0.986111, .75R: 0.833333, count: 72, class_loss = 2.380716,
 total_bbox = 4965976, rewritten_bbox = 0.072997 %
Saving weights to backup//yolov4_training_2000.weights
Saving weights to backup//yolov4_training_last.weights
Saving weights to backup//yolov4_training_final.weights
```

Figure 4: Model on Google Colab.

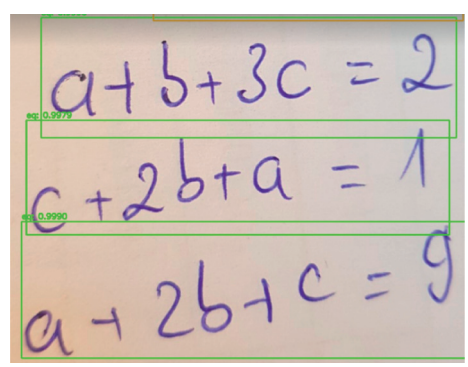


Figure 5: Result of training data.

contains the equation. We will then identify each of the equations individually. The results are shown in Figures 6 and 7.

(ii) Step 2: processing equation area. We will proceed to process the area of the equation with white characters and a black background. If the equation area is out of bounding box, we will fill in black color. We will use OpenCV functions such as "cv2.cvtColor, cv2.bitwise_not, cv2.threshold" to perform tasks. The results are shown in Figures 8 and 9.

(iii) Step 3: rotating image. Steps are performed as follows:

(1) Determining rectangle with the smallest area surrounding character and rotation angle
(2) Finding matrix to rotate image
(3) Performing to rotate image

To define a rectangle with the smallest area around a character and rotation angle, we will first define all characters with valuable pixel more than zero. We will then use function "cv2.minAreaRect" to determine the rotation angle (θ) that is depicted in Figures 10 and 11.

After getting the rotation angle of the equation, we calculate the center coordinates of the original image. Therefore, we will calculate the matrix to rotate the image according to [33]. The matrix is obtained when using rotation angle (θ) and center coordinates of image.

If we have an image of length as h and width as w, the center coordinate of image is as follows:

$$\begin{aligned} x &= w//2, \\ y &= h//2. \end{aligned} \tag{1}$$

We can represent transformation matrix as follows:

$$T = \begin{bmatrix} \alpha & \beta & (1-\alpha)x - \beta y \\ -\beta & \alpha & \beta x - (1-\alpha)y \end{bmatrix}, \tag{2}$$

where $\alpha = \text{scale}\cos(\theta)$, $\beta = \text{scale}\sin(\theta)$, and θ is rotation angle. In the paper, we choosescale = 1.

The image transformation can be represented by an affine transformation. The matrix can be used to represent rotation, translation, and scaling operations. The usual way to represent an affine transformation is to use a 2×3 matrix. We have the matrix M calculated as follows:

$$M = \begin{bmatrix} a_{00} & a_{01} & b_{00} \\ a_{10} & a_{11} & b_{10} \end{bmatrix}. \tag{3}$$

2D image vector is as follows:

$$X = \begin{bmatrix} x \\ y \end{bmatrix}. \tag{4}$$

2D image vector after transformation is as follows:

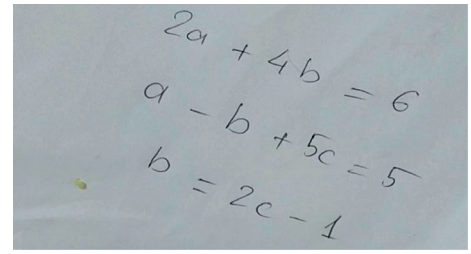


Figure 6: An example of input image.

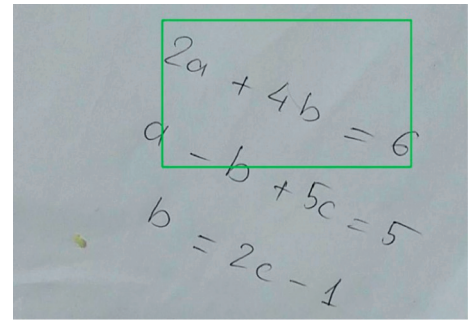


Figure 7: Equation area recognition.

$$\begin{aligned} T &= M.[x, y, 1]^T, \\ T &= \begin{bmatrix} a_{00}x + a_{01}y + b_{00} \\ a_{10}x + a_{11}y + b_{10} \end{bmatrix}. \end{aligned} \tag{5}$$

To save time, we perform the steps of matrix calculation and use two functions "cv2.getRotationMatrix2D, cv2.warpAffine". Results are shown in Figure 12.

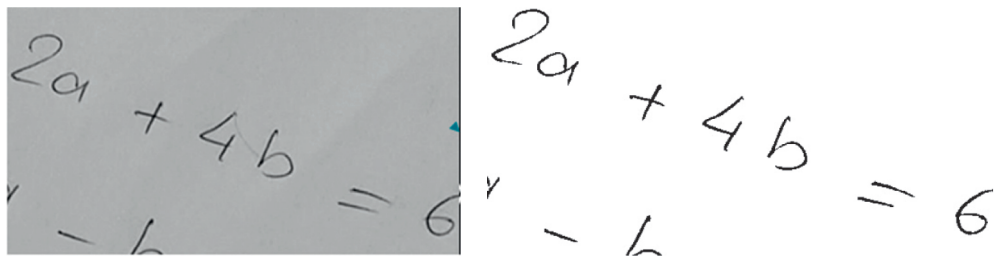
3.2.3. Equation Recognition. After rotating the image, we will detect the equation again. The purpose of this step is to identify equations more accurately. Results are shown in Figure 13.

3.2.4. Character Recognition. In this step, we also perform similar to step 1 to identify equation. The number of images to train is 285.

3.2.5. Character Classification. Implementation steps are as follows:

(i) Step 1: preparing the data. We will prepare 20-character data as shown in Figure 14 and Table 2. The images in the dataset are black and white images where the foreground and background are white and black, respectively.

The model to train data is VGG-16 that is proposed by [5]. 13 layers of convolution2D and 3 layers fully



(a) (b)

Figure 8: Equation area (a) before and (b) after processing.

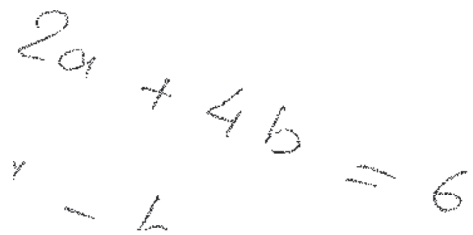


Figure 9: Equation area recognition.

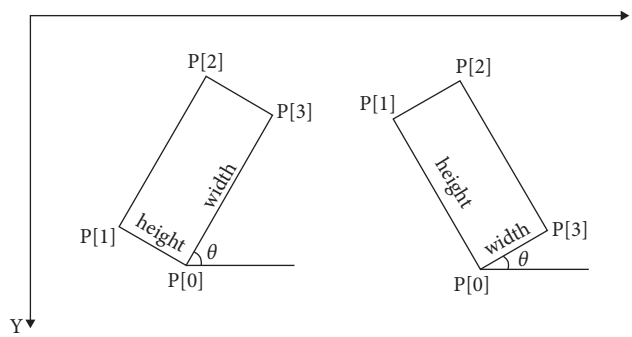


Figure 10: Describing of angle θ of rectangle.

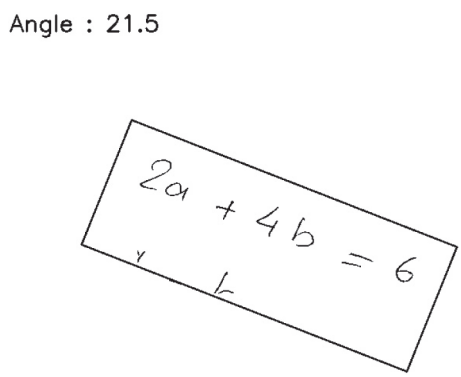


Figure 11: Defining rectangle around character area.

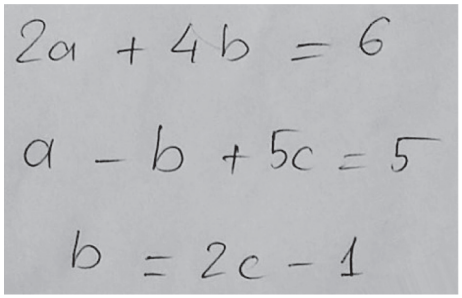


Figure 12: The original image after rotating.

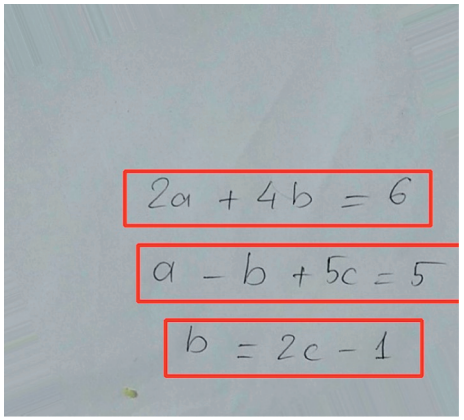


Figure 13: Image after rotating and identifying.

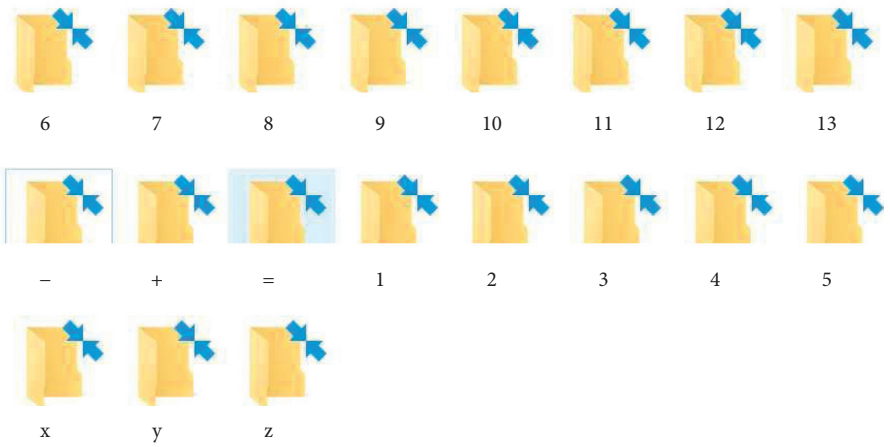


Figure 14: An example of character dataset.

Table 2: Number of images of each letter to be prepared for training.

No.	Characters	Number of images
1	—	2177
2	+	2199
3	=	2142
4	0	2159
5	1	2255
6	2	2249
7	3	2248

Table 2: Continued.

No.	Characters	Number of images
8	4	2233
9	5	2270
10	6	2233
11	7	2174
12	8	2221
13	9	2179
14	a	2068
15	b	2220
16	c	2168
18	d	2145
19	x	2177
20	y	2212
21	z	1891

connected as shown in Figure 15 with image input of 28×28 pixels.

Features of the VGG-16 architecture are used as follows:

(1) Input layer: the VGG-16 architecture receives 28×28 input image with three color channels as red, green, and blue.
(2) Convolution layer: image goes through 13 convolution layers where the size of the kernels is 3×3 with stride of 1, and the output size after passing through the convolution layer remains equal.
(3) After going through convolutional layer, we will use Batch to normalize data.
(4) Trigger function: the trigger function is used by ReLU.
(5) Pooling class: MaxPooling is used in the VGG-16 architecture with size 2×2 and equal stride. After 2 or 3 consecutive convolution layers, there will be a MaxPooling layer.
(6) Fully connected class: the first class has 512 nodes, and the last one has 20 nodes corresponding to the 20 character labels that we want to classify.
(7) Total number of parameters is 14,932,692 where learning and remaining parameters are 14,924,244 and 9,448, respectively.

(ii) Step 2: performing training. Next step is training process using the VGG-16 model. The results are shown in Figure 16.

(iii) Step 3: performing classification. In this step, we will perform to sort data. Results are shown in Figure 17.

3.2.6. Processing Text and Solving Equation. We get results while combining to recognize equation and character, as shown in Figure 18.

We will use the SymPy module to solve the equation after extracting text from the image. Its goal is to be a fully featured computer algebra system while keeping the code as simple as possible for easy understanding and extensibility.

3.2.7. Deploying on Website. In the final step, we deploy the program for the website application. We use the Flask framework that is easy to implement.

4. Simulation and Result

4.1. Setup. In our simulation, we setup parameters as follows. The VGG-16 architecture receives 28×28 input image with three color channels as red, green, and blue. Convolution has 13 layers where the size of the kernels is 3×3 with stride of 1. We use a trigger function as ReLU. MaxPooling is used in VGG-16 architecture with size 2×2. The first fully connected class has 512 nodes, and the last fully connected class has 20 nodes. The total number of parameters is 14,932,692 where the learning parameters are 14,924,244.

We use 23 images for testing and evaluate two parameters (accuracy of equation and character and time processing). We use the Tesla P100-PCIE-16 GB GPU for training and testing data.

4.2. Result. Our system can recognize equations up to 91.3% with a 23-image test image dataset. In the case of character recognition and classification, the accuracy of our system is up to 97%. Results are shown in Figures 19–22.

In Figure 19, results show accuracy of up to 100% in both first and second order equations. In Figure 20, we perform a loss function evaluation according to the YOLOV4 model. Results show that the loss function decreases when the number of iterations increases. When the number of iterations gains 2000, the average loss function is 3.5% for 0.16 hours.

Figures 21 and 22 are the results of loss and accuracy functions for the VGG-16 model. The results show that both functions will reach saturation by 10 epochs. The results show that proposal model is highly feasible, especially for teaching and training.

Table 3 shows the accuracy result for the calculation. In Table 3, we see that the VGG-16 model gains accuracy 2% higher than the YOLOV4 model.

Table 4 shows accuracy result for calculation when we perform evaluations for 23 different equations. In Table 4,

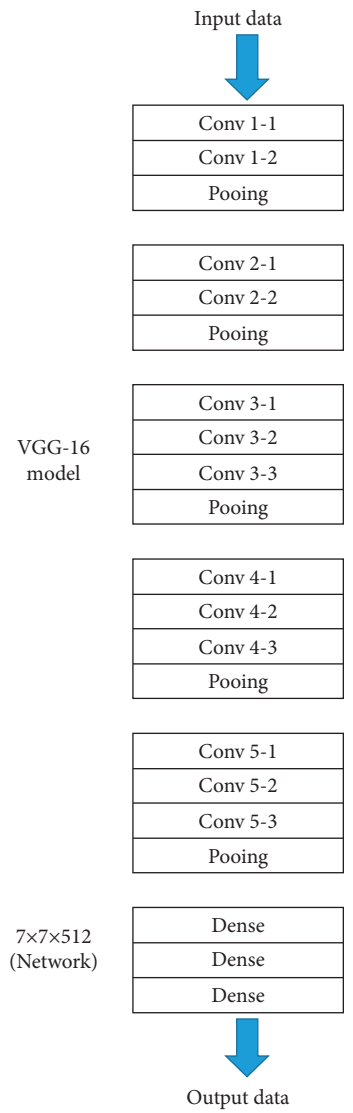


Figure 15: Flowchart of training the VGG-16 model based on [5].

```
Epoch 00003: val_acc improved from 0.31972 to 0.53807, saving model to /content/drive/My Drive/Final Project/Khang/weights/model_ocr_test.h5
Epoch 4/40
1090/1090 [==============================] - 80s 74ms/step - loss: 1.1847 - acc: 0.5838 - val_loss: 0.7010 - val_acc: 0.7525

Epoch 00004: val_acc improved from 0.53807 to 0.75252, saving model to /content/drive/My Drive/Final Project/Khang/weights/model_ocr_test.h5
Epoch 5/40
1090/1090 [==============================] - 79s 73ms/step - loss: 0.7401 - acc: 0.7501 - val_loss: 0.3086 - val_acc: 0.9169

Epoch 00005: val_acc improved from 0.75252 to 0.91686, saving model to /content/drive/My Drive/Final Project/Khang/weights/model_ocr_test.h5
Epoch 6/40
1090/1090 [==============================] - 80s 73ms/step - loss: 0.4089 - acc: 0.8975 - val_loss: 0.2168 - val_acc: 0.9435

Epoch 00006: val_acc improved from 0.91686 to 0.94346, saving model to /content/drive/My Drive/Final Project/Khang/weights/model_ocr_test.h5
Epoch 7/40
1090/1090 [==============================] - 80s 74ms/step - loss: 0.2796 - acc: 0.9327 - val_loss: 0.1775 - val_acc: 0.9530

Epoch 00007: val_acc improved from 0.94346 to 0.95298, saving model to /content/drive/My Drive/Final Project/Khang/weights/model_ocr_test.h5
Epoch 8/40
1090/1090 [==============================] - 80s 73ms/step - loss: 0.2326 - acc: 0.9454 - val_loss: 0.1641 - val_acc: 0.9569

Epoch 00008: val_acc improved from 0.95298 to 0.95688, saving model to /content/drive/My Drive/Final Project/Khang/weights/model_ocr_test.h5
Epoch 9/40
1090/1090 [==============================] - 80s 73ms/step - loss: 0.1914 - acc: 0.9543 - val_loss: 0.1522 - val_acc: 0.9600

Epoch 00009: val_acc improved from 0.95688 to 0.95998, saving model to /content/drive/My Drive/Final Project/Khang/weights/model_ocr_test.h5
Epoch 10/40
 320/1090 [=======>......................] - ETA: 53s - loss: 0.1640 - acc: 0.9595
```

Figure 16: Data training process using the VGG-16 model.

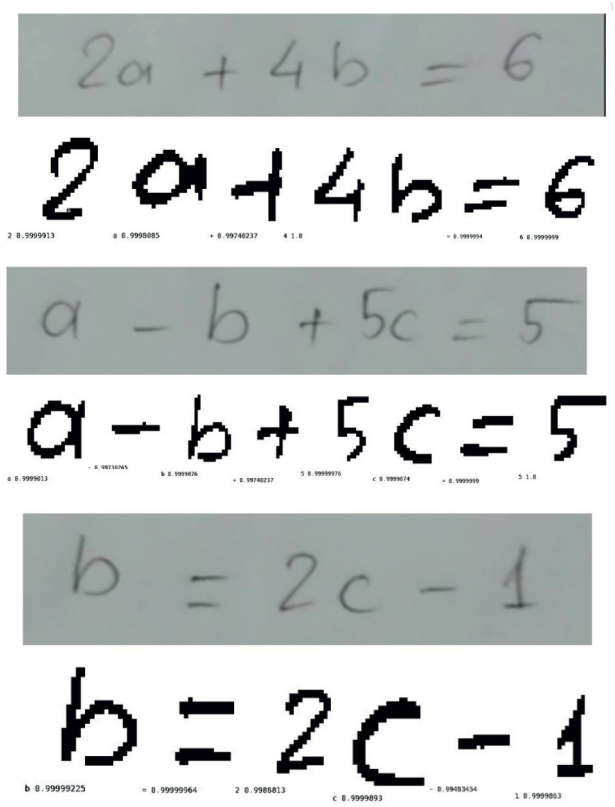


Figure 17: Result of character classification.

```
2x+4=6 → 2*x + 4 = 6
3x*4-2x+5 → 3*x**4 – 2*x +5
3x4y → 2*x**4*y
```

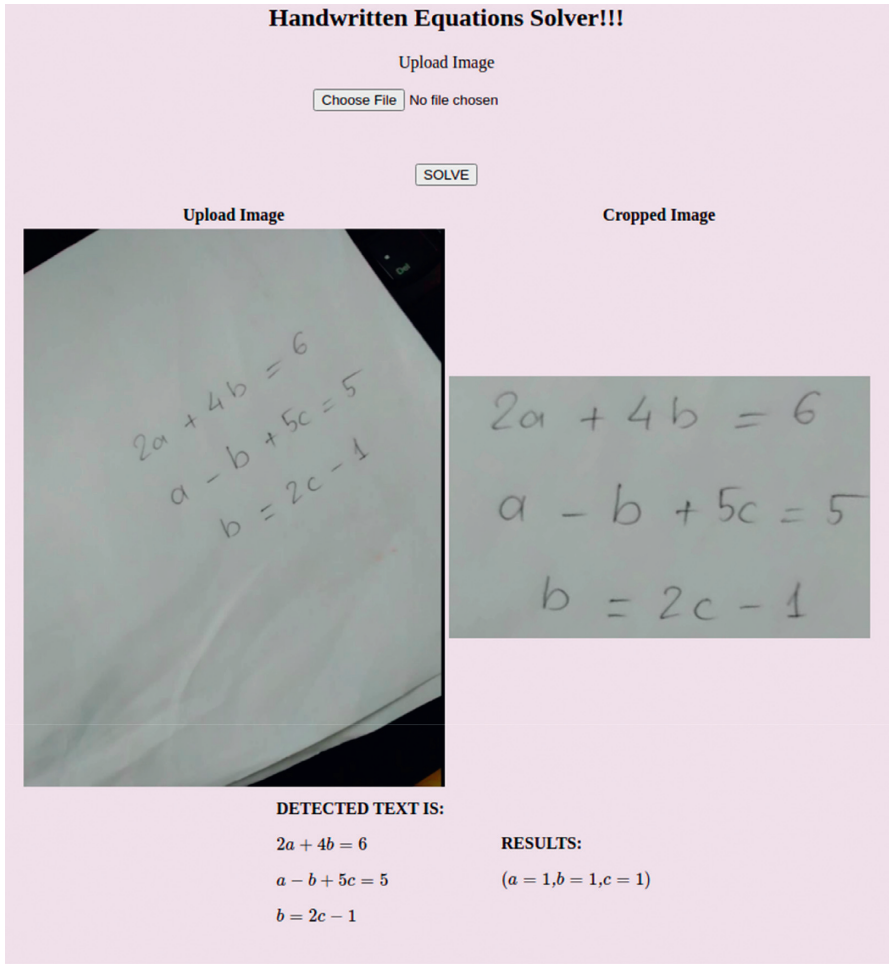
Figure 18: An example of text processing before using the SymPy module.

results show that the recognition rate is 96% with 22 correct cases and only 1 of wrong recognition.

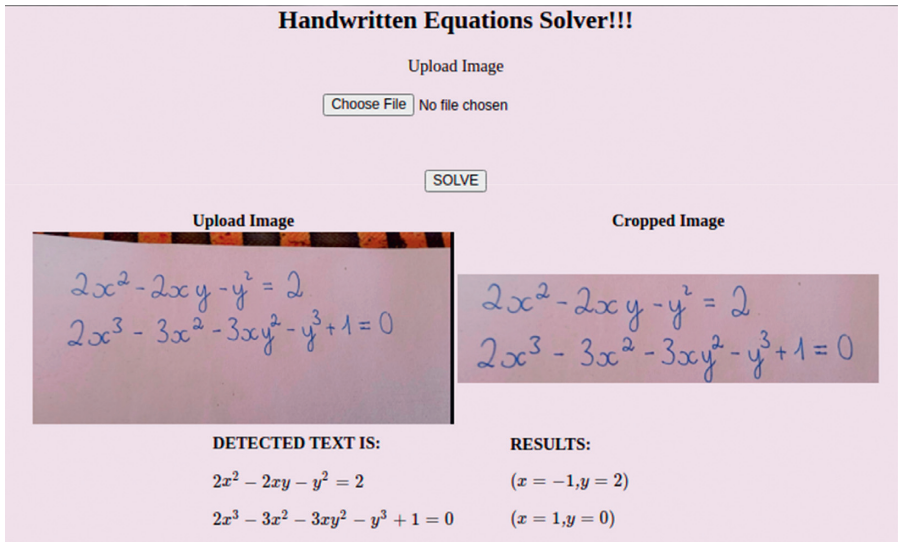
We see that the processing time per equation is 14 seconds on Google Colab with a Tesla P100-PCIE-16 GB GPU.

To compare with the traditional method, we can easily use hands to calculate. When encountering problems with high-order equations or many parameters, it is difficult to find a solution. Therefore, a proposal system with a processing time of up to 14 seconds is able to help students or researchers quickly find a solution that is useful for smart applications.

Besides, we compare the accuracy of the proposal with other methods. The results are shown in Table 5 that the proposed method has the best results in terms of handwriting recognition and equation solving with an accuracy of up to 95%.



(a)



(b)

Figure 19: Deploying on the website using Flask for two cases: (a) low equation and (b) high equation.

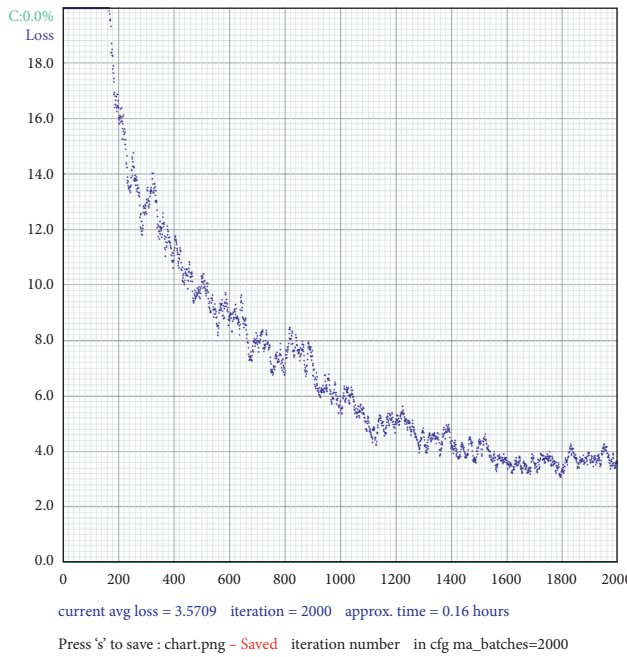


Figure 20: Result of loss function while training YOLOV4.

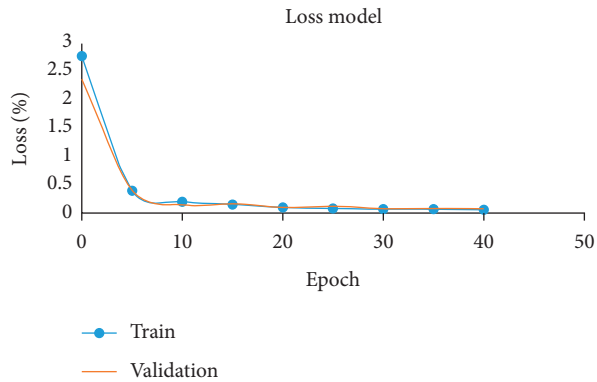


Figure 21: Result of loss function while training VGG-16.

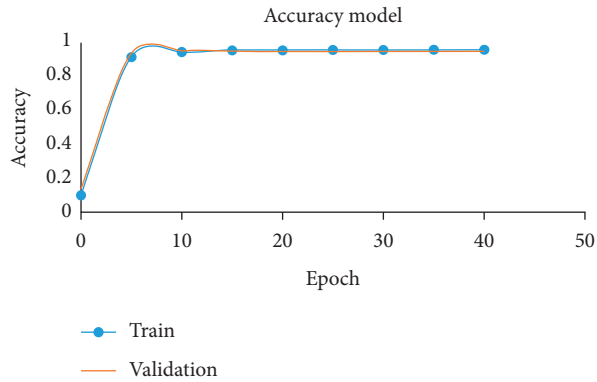


Figure 22: Result of accuracy function while training VGG-16.

Table 3: Comparing accuracy of model after training.

Model	Accuracy of model (%)
Yolov4 for equation	96
Yolov4 for character	95
VGG-16	98

Table 4: Number of images of each letter to be prepared for training.

No. input image	Total equations	Total number of characters	Correct number of characters	Number of correct equations	Correct character ratio	Rate equations correct
Image 1	2	16	16	2	1	1
Image 2	3	21	20	3	0.95	1
Image 3	3	21	21	3	1	1
Image 4 ÷ 21	3	23	22	3	0.95	1
Image 22	2	30	25	0	0.83	0
Image 23	1	14	13	2	0.92	1

Table 5: Comparing accuracy of proposal with other methods.

No.	Method	Average correct character ratio (%)	Average rate equations correct (%)	Average accuracy of method (%)
1	[20]	91.08	39.11	65,09
2	[25]	—	—	90.38
3	[10]	—	—	78.1
4	[23]	85.7	86.1	85.9
5	**Proposal (YLOV4 + VGG-16)**	**94.78**	**95.65**	**95.21 = 1/2 * (94.78 + 95.65)**

Our proposal has been shown in bold.

5. Conclusion

The paper focuses on the studding of neural networks for identifying and solving systems of equations. In this paper, we have identified equations with an accuracy of 97% and recognized and classified characters up to 95%. However, the system also has the disadvantage that processing time is not fast since it has to go through many models.

Therefore, we will change the system by removing models and collecting more data about equations and characters to improve the accuracy of the system in the future.

Data Availability

The authors confirm that all the data in the study were built by themselves. They used 23 images for training and testing. They also used the Tesla P100-PCIE-16GB GPU to perform simulation.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research was carried out in the framework of the project funded by the Ministry of Education and Training (MOET), Vietnam, under the grant B2020-BKA-06. The authors would like to thank the MOET for their financial support.

References

[1] A. Bochkovskiy, C.-Y. Wang, and H.-Y. Liao, "Yolov4: optimal speed and accuracy of object detection," no. 1–17, 2020.

[2] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," in *Proceedings of the 2017 IEEE International Conference on Image Processing (ICIP)*, pp. 3645–3649, Beijing, China, September 2017.

[3] C. Kumar, R. Punitha, and Mohana, "Yolov3 and yolov4: multiple object detection for surveillance applications," in *Proceedings of the 2020 Third International Conference on Smart Systems and Inventive Technology (ICSSIT)*, pp. 1316–1321, Tirunelveli, India, August 2020.

[4] S. Canu, "Train yolo to detect a custom object (online with free gpu)," 2020, https://pysource.com/.

[5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[6] Ø. Due Trier, A. K. Jain, and T. Taxt, "Feature extraction methods for character recognition-a survey," *Pattern Recognition*, vol. 29, no. 4, pp. 641–662, 1996.

[7] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3642–3649, Providence, RI, USA, June 2012.

[8] S. S. Gharde, P. V. Baviskar, and K. P. Adhiya, "Identification of handwritten simple mathematical equation based on svm and projection histogram," 2013.

[9] C. Lu and K. Mohan, "Recognition of online handwritten mathematical expressions using convolutional neural networks," pp. 1–7, 2015.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[11] C. J. Egelhoff, D. M. Blackketter, and J. L. Benson, "Algorithms for solving nonlinear equation systems assist students to become better problem solvers," in *Proceedings of the FIE'99 Frontiers in Education. 29th Annual Frontiers in Education Conference. Designing the Future of Science and Engineering Education. Conference Proceedings (IEEE Cat. No.99CH37011*, vol. 1, pp. 12A4/17–12A4/22, San Juan, PR, USA, November 1999.

[12] K. Yamamura and R. Watanabe, "Finding all solution sets of piecewise-linear interval equations using integer programming," in *Proceedings of the 2017 European Conference on Circuit Theory and Design (ECCTD)*, pp. 1–4, Catania, Italy, September 2017.

[13] C. Liu, L. Zuo, X. Li, and X. Tian, "An improved algorithm for identifying mathematical formulas in the images of pdf documents," in *Proceedings of the 2015 IEEE International Conference on Progress in Informatics and Computing (PIC)*, pp. 252–256, Nanjing, China, December 2015.

[14] H. Yu and R. V. Iyer, "On the solution of operator equation problems with application to preisach density estimation," in *Proceedings of the 2016 American Control Conference (ACC)*, pp. 2427–2432, Boston, MA, USA, July 2016.

[15] A. Blanco, M. Delgado, and I. Requena, "Solving fuzzy relational equations by max-min neural networks," in *Proceedings of the 1994 IEEE 3rd International Fuzzy Systems Conference*, vol. 3, pp. 1737–1742, Orlando, FL, USA, June 1994.

[16] O. L. Men, Z. Ismail, and M. Abidin, "Using maths model method in solving pre-algebraic problems among year five students," in *Proceedings of the 2018 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*, pp. 222–227, Wollongong, NSW, Australia, December 2018.

[17] A. Qian and Y. Gui, "Determining an unknown source in the heat equation by a mollification regularization method," in *Proceedings of the 2010 International Conference on Computational Intelligence and Software Engineering*, pp. 1–4, Wuhan, China, December 2010.

[18] J. Zhang, H. Gao, and G. Han, "A stable algorithm for parameter identification in heat conduction equations," in *Proceedings of the 2017 IEEE 17th International Conference on Communication Technology (ICCT)*, pp. 1809–1812, Chengdu, China, October 2017.

[19] C. L. Karr, A. Banerjee, and P. Mishra, "Solving an inverse partial differential equation for a two dimensional heat conduction problem with oscillating boundary conditions using an artificial immune system," in *Proceedings of the 2004 International Conference on Machine Learning and Applications*, pp. 99–106, Louisville, KY, USA, Decmber 2004.

[20] M. B. Hossain, F. Naznin, Y. A. Joarder, M. Zahidul Islam, and M. J. Uddin, "Recognition and solution for handwritten equation using convolutional neural network," in *Proceedings of the 2018 Joint 7th International Conference on Informatics, Electronics Vision (ICIEV) and 2018 2nd International Conference on Imaging, Vision Pattern Recognition (icIVPR)*, pp. 250–255, Kitakyushu, Japan, June 2018.

[21] C. Ramteke, T. Chakrabarti, B. Sarangi, and R. A. Pandey, "Synthesis of silver nanoparticles from the aqueous extract of leaves of ocimum sanctum for enhanced antibacterial activity," *Journal of Chemistry*, vol. 2013, no. 1–7, 2012.

[22] J. Pradeep, E. Srinivasan, and S. Himavathi, "Diagonal feature extraction based handwritten character system using neural network," *International Journal of Computer Application*, vol. 8, p. 10, 2010.

[23] N. Kushman, L. Zettlemoyer, R. Barzilay, and Y. Artzi, "Learning to automatically solve algebra word problems," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, pp. 271–281, Baltimore, MA, USA, June 2014.

[24] S. Roy and D. Roth, "Solving general arithmetic word problems," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 1743–1752, Lisbon, Portugal, September 2015.

[25] A. Mitra and C. Baral, "Learning to use formulas to solve simple arithmetic problems," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pp. 2144–2153, Berlin, Germany, August 2016.

[26] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: unified, real-time object detection," in *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779–788, Las Vegas, NV, USA, June 2016.

[27] W. Liu, D. Anguelov, D. Erhan et al., "SSD: Single shot MultiBox detector," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., Springer International Publishing, Manhattan, NY, USA, pp. 21–37, 2016.

[28] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 580–587, Columbus, OH, USA, June 2014.

[29] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017.

[30] F. Chollet, *Deep Learning with Python*, Manning Publications Co., Shelter Island, NY, USA, 1st ed. edition, 2017.

[31] S. Gollapudi, *Learn Computer Vision Using OpenCV: With Deep Learning CNNs and RNNs*, APress, 1st ed. edition, 2019.

[32] A. Rosebrook, *Practical Python and OpenCV + Case Studies: An Introductory, Example Driven Guide to Image Processing and Computer Vision*PyImageSearch, 2016, https://books.google.com.vn/books?id=ZoH0xwEACAAJ.

[33] C. P. Brokate, *Image Rotation Using OpenCV*, https://cristianpb.github.io, 2017.