IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2020.DOI

Graph-Based Locality-Sensitive Circuit Sketch Recognizer

SONGHUA LIU, LINFENG LI, FUKANG JIU, GUIHUAN FENG
State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
Corresponding author: Guihuan Feng (e-mail: fenggh@smail.nju.edu.cn).

This work was supported in part by the National Key Research and Development Project under Grant 2018YFB1004900.

ABSTRACT The understanding of circuit diagram is very important for the study of electrical engineering. Existing circuit diagram simulation tools are mostly based on GUI interface and rely on users to click or drag icons with mouse, which requires them to be familiar with the software and distracts a great deal of their attention from the circuit diagram itself. Although a lot of previous works have devoted to designing algorithmic solution to recognize hand-drawn circuit diagrams automatically, there still exists strict constraints on users' drawing habits and stroke orders. In order to address these inconveniences, this paper proposes a novel sketch recognition algorithm named LS^4D. It uses graph to model the input strokes and their relationships, and leverages cycles by local strokes to detect some circuit components. Theoretical derivations have demonstrated that LS^4D can efficiently recognize diagrams with different drawing styles and arbitrary stroke orders. To furthermore illustrate the practical value of the proposed approach, we construct a prototype of pen-based circuit diagram system based on LS^4D, which enables users to draw circuit diagrams directly on the digital screen without any other restriction. An experiment of 158 samples collected from 17 users is conducted on the designed platform. Our approach has achieved 93.04% recognition accuracy and overall 4.53 from a 5-scale user satisfaction rating, which outperforms previous state-of-the-art methods numerically. It is shown that the same approach can also be generalized to many other sketch recognition applications with minor modifications. To facilitate future researches and applications, we publish our source code, model, and training data at https://github.com/Huage001/Graph-Based-Circuit-Painter.

INDEX TERMS Pen Interaction, Graph, Circuit Sketch Recognition

I. INTRODUCTION

Circuit diagram is the basis and difficulty of electricity in physics and engineering. With modern technologies, circuit diagrams can be simulated on screens with users operating positions and relationships of a series of components. Users especially students can get the visualized results from such simulators and understand electrical rules and laws gradually with the assistance of interactive programs, which also benefits teaching in education. A circuit simulation system should be user-friendly, $i.e.$, users can input to the system conveniently, and understandable enough for users to learn the internal electrical principles. At present, most existing public circuit simulation systems require users to drag components to proper positions with mouse, $e.g.$, [1]. This approach is easy to design yet relatively complex and tedious for users since these operations can distract user's attention from the circuit diagram itself significantly and fail to provide an immersive drawing and learning environment. Therefore, this paper designs a pen-interaction circuit diagram recognition and analysis prototype system. It allows users to draw circuit diagrams on a tablet and is as convenient as drawing on paper.

In circuit sketch recognition task, a core issue is to conduct stroke grouping or segmentation, $i.e.$, to tell which strokes belong to a specific circuit component. Due to the lack of visual nervous system like human brain, it is not as easy as it sounds for computers. There are a number of works focusing on this direction. However, all these methods have strict restrictions on stroke orders. For example, Gennari $et\ al.$ [2] proposed to enumerate different stroke combinations with some heuristic pruning strategies. Nevertheless, to ensure the time complexity over the number of strokes is not exponential, they had to make a constraint that all the circuit components must be drawn one by one without interspersing ($e.g.$, drawing a stroke for one component, then turning to another component, and finally going back to finish the previous component). Methods like [3], [4] and

[5] achieve stroke segmentation by training and inference schema with dynamic Bayes networks, $i.e.$, learning from prior knowledge about how a specific component was drawn. On one hand, it is inconvenient to build a proper database for stroke segmentation training. On the other hand, it is hard to ensure that the model works for all possible stroke orders, especially when it comes across a case that it has never seen in training stage. Some recent deep learning methods like [6], [7], [8], and [9], which perform well on image object detection and semantic segmentation tasks, are not suitable for this scenario since they need even larger labeled datasets to train a reliable deep neural network and it is hard to guarantee that the model can deal with people's different drawing habits.

This work aims at handling sketches drawn with all possible stroke orders. Focusing on spacial information of input strokes, we use graph models to represent users' input sketches. And then local cycles in graphs are utilized to detect and localize circuit components as well as analyze their connection relationships. Since temporal orders or time information of strokes is not taken into consideration, it is shown that our algorithm is not sensitive to stroke orders or drawing habits. Meanwhile, a pen-interaction circuit recognition and analysis system is developed based on the proposed model and algorithm. Theoretical guarantee and experiments have shown that our algorithm and system are efficient as well as stable.

This paper has mainly three contributions:

1 proposes an efficient representation model and algorithm for circuit recognition task supporting arbitrary stroke orders;
2 designs a complete, stable, and satisfying prototype system to validate the superiority of the algorithm, which can be applied to circuit diagram teaching and learning;
3 makes up for the deficiencies of open-source code and datasets in the areas of sketch recognition and circuit component classification.

The following parts will be organized as follows: section II introduces some related works about this paper; section III elaborates core models and methods proposed for circuit sketch recognition task; section IV gives theoretical analysis to the correctness and time complexity to our main algorithm; system architecture, implementing details, and our experimental study, which are used to verify the efficiency of our algorithms in practice, are shown in section V; and section VI concludes this paper and introduces some future works.

II. RELATED WORKS

Circuit sketch recognition is not a new research topic since decades ago. Nevertheless, each previous approach has its own limitations.

Firstly, there are some arts operating sketch image directly and using pixel information to analyze connection relationships between components. For instance, Edwards $et\ al.$ [10] took advantage of the concept of *ink density* (density of painted pixels) to identify the occurence of components and connecting points. When the ink density is larger than a pre-defined threshold, it is assumed that there is a circuit component or connecting structure in this area. Similar ideas were also adopted in [11], [12], and [5]. One major concern of these methods is that they require a completed sketch image and is weak to recognize the circuit online. Besides, since the diversity of sketch drawing habits, such ink density strategy is not robust enough to handle irregularly drawn sketches, $e.g.$, with some mistakenly touching noises. In addition to ink diversity, Gennari L $et\ al.$ [2] also combined the geometric features of the image and domain knowledge to explain circuit sketches. However, their approach fails to make full use of spacial features of sketch, but relies on enumerating all possible time-continuous stroke combinations as first-hand candidates inflexibly. De Silva $et\ al.$ [13] used this approach to build a circuit educational system that enables students to write circuit equations and compare the answers with those given from the system analysis results. Their work also remains the same concerns mentioned above.

There are also works dealing with online sketches. For this representation, a key issue is to sort input strokes into their belonged components or specify which component each stroke belongs to. In early stage, this classification was controlled by users manually. For example, Fonseca $et\ al.$ [14] divided strokes by pause, $i.e.$, if the user pauses for a certain time interval before drawing next stroke, the previous strokes would belong to a component and next strokes would belong to another. Liwicki $et\ al.$ [15] extracted independent circuit components by switching modes. These methods can certainly work yet not flexible and user-friendly enough, since they highly rely on cooperation of users.

Many studies have proposed automated solutions to this inconvenience. Valois $et\ al.$ [16] completed the identification and beautification of hand-drawn circuit diagrams by extracting the structure and topological relationship of the images. Their approach views the stroke group that has the highest recognition (classification) confidence value as a true grouping or segmentation result, which high relies on the performance of circuit symbol classifier, and is hard to deal with the inaccuracy of sketches [17], since some minor shifts can result in fluctuation of the confidence value. Dreijer $et\ al.$ [18] proposed a novel normalization process to make it easier to recognize components; Alvarado $et\ al.$ [3] utilized Bayes nets to infer the relationships between each strokes according to stroke temporal information. However, in these methods, users are not allowed to draw other parts of a circuit diagram before finishing the current component, which is not always the case actually. Besides, they require developers to collect and label a number of training samples before the system is ready to work. Latter on, Sezgin $et\ al.$ [4] extended the probability graph model [3] to support interspersed drawings. Nevertheless, similar to [5], it is still influenced by stroke order to a large extend, since it is a learning-based approach and it is difficult to get a stroke order that never appears in the training set right.

Feng $et\ al.$ [17] used two-dimensional dynamic program-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3035545, IEEE Access

Works	Main Model or Algorithm	Information Relied on	Supervised Training[1]	Online
Edwards *et al.* [10]	Ink Density	Ink Density	No	No
Gennari L *et al.* [2]	Enumerate with Pruning	Temporal and Spacial Order	No	No
Fonseca *et al.* [14]	Pause-Based Segmentation	Temporal Order	No	Yes
Liwicki *et al.* [15]	Mode-Based Segmentation	Temporal Order	No	Yes
Valois *et al.* [16]	Highest Confidence Value	Temporal and Spacial Order	No	Yes
Alvarado *et al.* [3]	Dynamic Bayes Network	Temporal Order	Yes	Yes
Sezgin *et al.* [4]	Dynamic Bayes Network	Temporal Order	Yes	Yes
Altun *et al.* [5]	Dynamic Bayes Network	Temporal Order	Yes	Yes
Feng *et al.* [17]	2-D Dynamic Programming	Spacial Order	No	Yes
Ours	Graph Model	Spacial Order	No	Yes

TABLE 1. Summary of related works.

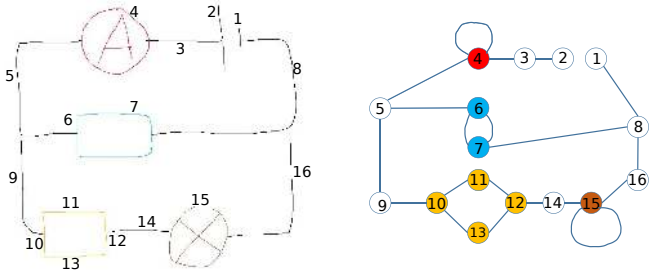


FIGURE 1. We model users' input strokes with graph. Left: user's sketch. Right: the corresponding graph model.

Component	Sketch	Component	Sketch
Light Bulb		Ammeter	
Motor		Voltmeter	
Resistor		Diode	
Buzzer		Transistor	
Power Supply		Capacitor	

TABLE 2. Circuit component samples used in this paper.

ming to store the information of each state of each stage of the user's drawing of the circuit diagram to identify the circuit diagram and took spacial information into consideration. This method relaxes stroke order restriction for users by considering temporal and spacial information of input strokes at the same time. However, to reduce search space, they have to set a upper bound that controls the maximal number of interspersed strokes, $i.e.$, when there are more strokes than this threshold between two strokes that belong to a same component, the algorithm would discard it and result in a recognition failure. In addition, in terms of running time, it takes nearly one minute to process a regular circuit diagram.

There is a intuitive summary of these previous works as well as this work in Tab.1. Different from previous works, this paper proposes an $\mathcal{O}(n)$ algorithm, where n is the total number of strokes. It can support arbitrary stroke order as well as accurately detect and extract sketch circuit components with a novel graph model. Furthermore, we also build a prototype system that benefits teaching and learning for electrical knowledge. It is shown that our method can be applied to other fields with minor changes.

III. METHODS

This section presents the core model and algorithm used in circuit sketch recognition task in this paper. We use graph to model users' input strokes and it runs in an online scenario. Each stroke is represented as a vertex in the graph. There would be an edge between two vertices if and only if their corresponding strokes come into contact with each other, $i.e.$, there is a common pixel on one of strokes' endpoint on the screen. One advantage of using such graph model is that it is an abstract representation and immune to users' different sketch habits geometrically. As shown in Fig.1, every stroke in the left part corresponds to its own unique vertex in the right part, which is similar to the concept of dual graph. It is worth noting that edge set of our graph model should be a multi-set since there can be more than one contacting point between two strokes, as shown in strokes 6 and 7 in Fig.1. In addition, we allow a stroke contacts with itself to form a selfloop in graph. As stroke 4 in Fig.1, the user sketches the border of the ammeter with only one stroke, with adjacent start point and end point forming a selfloop. Therefore, simple graph is not powerful enough to model our problem, so we use a more general graph with multiple edges and selfloops allowed.

With this graph model, we propose our main algorithm: Locality-Sensitive Special Sketch Symbol Detection (LS^4D). Our algorithm is designed based on following observations: most of circuit component symbols are equipped with a closed border, as shown in Tab.2 and [19], which are represented by cycles in graph models. Besides, strokes that form circuit components usually have closed connections in local spacial areas, which means that it is sufficient to only take strokes in the same local area into consideration. According to these two characteristics, we develop an algorithm that is good at detecting symbols with closed borders and name it Locality-Sensitive Special Sketch Symbol Detection (LS^4D). The main algorithm is shown in Algorithm 1, where s is new input stroke, G denotes the graph model mentioned in previous sections, V and E denote vertex and edge set of G respectively, S denotes previous stroke set, C denotes the set of cycles formed previously, *mark* means the kind of component a stroke belongs to, and *deg* means degree of a vertex. We firstly introduce the general LS^4D algorithm,

[1]Here the supervised training refers to whether it uses a training-and-inference manner for stroke segmentation or grouping, instead of circuit component classification.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3035545, IEEE Access

Algorithm 1 General LS^4D

Require: New stroke s; Graph model $G < V, E >$;
Previous stroke set S; Previous cycle set C.

1: Segment s into sub-strokes s_{n+1},s_{n+2},...,s_{n+m};
2: **for** $p = 1$ to m **do**
3: $mark(s_{n+p}) = -1$;
4: Update S and G with above modeling methods;
5: **if** $\exists c \in C$, satisfies $s_{n+p} \sqsubseteq c$ **then**
6: $c = c \cup \{s_{n+p}\}$;
7: **else**
8: $S_L \triangleq \{s_i | d_{i,n+p} \leq loc\}$;
9: $S_L = S_L - \{s_i | marked(s_i) \neq -1\}$;
10: $V_L = \{i | s_i \in S_L\}$;
11: $G_L = G[V_L]$;
12: $c = \{s_i | \exists j \in V_L, e_{i,j} \notin cut[G_L]\}$;
13: **if** $c \neq \emptyset$ **then**
14: $c = c \cup \{s_i | s_i \sqsubseteq c\}$;
15: $C = C \cup \{c\}$;
16: **end if**
17: **end if**
18: $r \triangleq classifier$(c's corresponding image);
19: $mark(s_i) = r, \forall s_i \in c$;
20: **end for**

which works for all sketch symbols ($e.g.$, symbols in circuit or flow chart sketch) with a closed border. Then to illustrate the effectiveness of LS^4D, we adopt circuit sketch recognition task as an example and apply the general algorithm with some minor adjustments.

A. GENERAL LS^4D

Firstly, we adopt a stroke segmentation step to process the new stroke in line 1 of Algorithm 1. In this step, we use both degree of the new stroke's corresponding vertex and cycle borders as indicators to divide the stroke. To be specific, as soon as user paints the first pixel, a new vertex would be allocated for this new stroke. Then we keep tracking user's nib. If it bumps into a stroke on a closed border, we would do segmentation here and start a new sub-stroke with a new vertex. It is worth noting that when a stroke collides to strokes other than this type, the algorithm would ignore it except that it happens on an endpoint of the new stroke or an existing stroke, to handle some crossing but non-joined wires. This kind of strategy makes crossing but not connected wires possible. In the segmentation process, when degree of the new stroke goes up to 2, the algorithm would also start a new sub-stroke, $i.e.$, the degree of vertex corresponding to a new sub-stroke is at most 2. It makes sure that the algorithm would not miss any required cycle, which is helpful to extract and analyze all the cycles one by one. Afterwards, we will focus on each sub-stroke, whose *mark* is -1 initially.

Based on the common feature of our target symbols, $i.e.$, a closed border, a natural idea is to divide the algorithm into two main branches to deal with the two cases respectively: the new stroke falls in a previously detected cycle and it is not in any cycle, corresponding to the condition statement in line 5 of Algorithm 1. We use notation $\sqsubseteq$ to indicate that a stroke falls in a closed border. Obviously, if a new stroke is in a cycle, it needs to send all strokes in this cycle (including the cycle itself) into the classifier module. Even though some of the strokes have already been classified, we have to send some marked strokes into classifier again to ensure correctness of final classification results, since the content in it has changed. See line 6 of Algorithm 1.

On the other hand, if the new stroke is not in any cycle, the algorithm would find its nearby strokes, whose corresponding vertices have a relatively short distance to the new vertex in the graph model, as shown in line 8 of Algorithm 1. Here we introduce a hyper parameter loc to indicate the locality sensitivity of the algorithm. Only vertices with a distance no more than loc to the new one are taken into account. Therefore, a higher loc means a weaker locality sensitivity. If there is a need to consider those strokes that are relatively far from the new one, loc should be higher. At the same time, the algorithm will consider less local information and more global information.

Since this branch satisfies the condition that the new stroke does not in any existing cycle, those marked strokes should not influence the segmentation and analysis of following strokes. Therefore, we do not take marked strokes into consideration and delete them from S_L in line 9 of Algorithm 1. We then extract the induced subgraph G_L for S_L's corresponding vertex set V_L in the current graph model G. After this step, the algorithm would detect the target symbols only in this subgraph G_L instead of the global graph, which reflects the local sensitivity of our algorithm. Subsequently, taking advantage of the closed border, we only need to examine whether there is a cycle in G_L. When a cycle is found, the algorithm can separate out the border strokes of target sketch symbols, and thus complete the detection tasks. In line 12 of Algorithm 1, the algorithm would find the cut-edge of G_L, which is equal to finding a cycle as shown by the following theorem [20].

Theorem 1. *An edge is a cut-edge if and only if it is not contained in any cycle.*

Considering that some users may draw the inner-frame content of a symbol at first and the border afterwards, in line 14 of Algorithm 1, we also add strokes which are enclosed in the extracted border, $i.e.$, the corresponding strokes of the cycle.

Finally, we send the final settled stroke set c's corresponding image into the classifier and mark these strokes with the classification result. Note that if the confidence level made by the classifier is not high enough, the system would view it as a negative sample. In this case, the classifier would return -1 as result. This mechanism prevents some meaningless circles and noise strokes to a large extend. At this time, the algorithm would process next sub-stroke. Upon all the sub-strokes are finished, the system would get ready to receive next input stroke.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3035545, IEEE Access

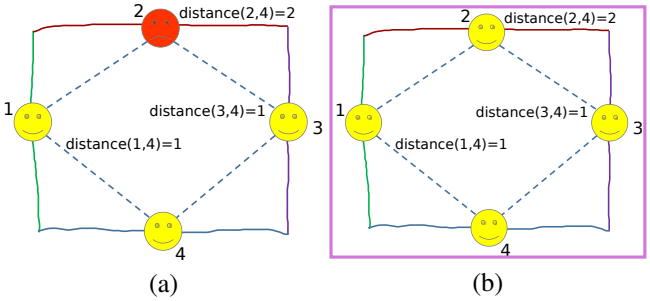


FIGURE 2. **Effect of the rectangular closure.**

Our algorithm has solid theoretical guarantees. Firstly, LS^4D can detect all the cycles meeting the requirements without omission, $i.e.$, the recall rate is 100%, as given in the following theorem:

Theorem 2. *LS^4D can detect all the cycles with length no more than $2 \times loc + 1$.*

Secondly, LS^4D can detect all the expected symbols without any error report, $i.e.$, the precision is 100%. Detailed proof of above theorem and conclusion can be found in the next section.

B. FINE-TUNED LS^4D FOR CIRCUIT SKETCH

Under the scenario of circuit sketch, according to our observation and user study, users tend to use relatively less strokes to finish drawing a symbol since borders of circuit symbols are relatively simple. From this point of view, we adjust the local sensitivity of the algorithm to the maximum, $i.e.$, set $loc = 1$. In other words, we only take those strokes that directly adjacent to the new stroke into consideration. Then the time complexity of the key operation of LS^4D for each sub-stroke is $\mathcal{O}(1)$ and total time complexity is $\mathcal{O}(|S|)$, which outperforms previous state-of-the-art techniques [2] [3] [4] [17]. Detailed analysis of time complexity can be found in the appendix section. Since the degree of a new input sub-stroke's corresponding vertex is at most 2, the algorithm can only find cycles with length no more than 3. Unfortunately, through our survey, some users still prefer to draw a resistor with 4 strokes, corresponding to the left, up, right, and bottom borders respectively. In this case, the basic LS^4D algorithm would fail to detect this kind of resistor, as shown in Fig.2(a). Therefore, we introduce the concept of rectangular closure and utilize it to fine-tune the basic LS^4D algorithm to solve this problem. To calculate the rectangular closure of a set of strokes, firstly, we need to find the smallest rectangle which can enclose all the strokes in the set, $i.e.$, the boundary coordinates of the rectangle is the maximum and minimum values of the strokes in the set on axes x and y (x_{max}, x_{min}, y_{max}, and y_{min}). Subsequently, we find all the strokes that are in the rectangle calculated in the last step. The set of these strokes is the rectangular closure of the original stroke set. In other words, we expand the original S_L through a rectangular rule and use expanded S_L instead of the original one for the following steps. Formally, the following two rectangular closure calculation steps are inserted after line 8 of Algorithm 1 to obtain fine-tuned LS^4D:

$$R \triangleq min\{Rect|\forall s_i \in S_L, s_i \sqsubseteq Rect\}, \tag{1}$$

$$S_L = \{s_i|s_i \in R\}. \tag{2}$$

As shown in Fig.2(b), the problem can be solved efficiently after we use the rectangular closure.

C. DISCUSSION

LS^4D is the core algorithm utilized to detect all the circuit symbols with a closed border in this paper. It does not need any labeled training data to perform the detection task. From this perspective, it is an unsupervised sketch object detection algorithm based on spacial information. Therefore, it can also be generalized to other detection and recognition scenarios concerning with sketches. For example, LS^4D can be applied to recognition of other type of sketches, such as flow charts and UML diagrams, with little modification. More generally, it is functional in a lot of computer-aid design (CAD) topics like sketch coloring based information extracted by the algorithm.

IV. THEORETICAL ANALYSIS

In this section, we supplement some materials to demonstrate the correctness of our main algorithm, $i.e.$, Graph-based LS^4D, from a theoretical point of view. And then, we analyze the time complexity of key steps for the algorithm. Thus, it is shown that our algorithm can complete the tasks efficiently with correctness.

A. PROOF OF ALGORITHM CORRECTNESS

1) Proof of Theorem 2 :

Proof. We use mathematical induction to prove that there is not any cycle with length no more than $2 \times loc + 1$ at any time step.

Initially, it is obvious that when there is only one stroke or one vertex, the theorem is true. Assume that the conclusion is true for all $n > 1$, where n denotes the size of vertex set of graph $G < V, E >$, $i.e.$, $|V|$. Suppose the algorithm is currently ready to process $n+1th$ sub-stroke s (here we only consider stroke that does not in any existing cycle) and the corresponding vertex is v.

Since we only consider those strokes with $mark = -1$, we can temporarily delete those vertices whose corresponding strokes do not satisfy this property and get the induced subgraph G'. Determined by the operation of LS^4D, we send strokes whose corresponding vertexes are on a cycle to classifier. According to the inductive hypothesis, there is no cycle with length no more than $2 \times loc + 1$ in G. As G' is a sub-graph of G, this hypothesis also applies to G'. Therefore, all the edges are either cutting edges, or on one or more cycles with length more than $2 \times loc + 1$ in graph G'.

Besides, due to our stroke segmentation method and pre-treatment operation on the stroke, the degree of a new sub-stroke's corresponding vertex is at most 2, $i.e.$, $deg(v) \leq 2$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3035545, IEEE Access

Obviously when $deg(v) = 1$ or $deg(v) = 0$, the new vertex is certainly not on any cycle so the inductive hypothesis is still true in these two cases. Therefore, if there are cycles with length no more than $2 \times loc + 1$ in graph $G' + v$, these cycles must contain vertex v at the same time and $deg(v) = 2$. The algorithm would consider vertices whose distance to the new vertex is at most loc so as long as the length of a cycle is no more than $2 \times loc + 1$, it would be sent to the classifier and get marked. In this way, the new vertex must be marked and there are less vertices that remain unmarked. Then the induced graph derived by these unmarked graph G'' is a subgraph of G'.

Since there is no cycle with length no more than $2 \times loc + 1$ in G', the same thing is also applied to G''. Hence, it is correct that there is not any cycle with length no more than $2 \times loc + 1$ at any time step for $n+1th$ sub-stroke and then the algorithm would detect and mark all the cycles with length no more than $2 \times loc + 1$, $i.e.$, the recall rate is 100%. □

2) Proof of Precision Rate :

Proof. Decided by the algorithm itself, our algorithm never sends a set of non-cycle strokes into the classifier. In other words, when we send a found cycle of the graph model into classifier, it is also a loop from the perspective of strokes, which can form the closed border of a symbol. Thus, the detection report of our algorithm is reliable.

In summary, LS^4D will detect all the required cycles accurately without omission. Since the target objects of the algorithm is symbols with closed borders, it can detect all the satisfied symbols correspondingly. □

B. ANALYSIS OF TIME COMPLEXITY

Due to the uncertainty of the number of sub-strokes, we take a single sub-stroke, instead of a whole input stroke as the unit for our time complexity analysis. According to the above analysis, the degree of a new vertex is at most 2. And if we set $loc = 1$, the algorithm can detect cycles whose length is at most 3, $i.e.$, the new vertex and its two adjacent vertices. Therefore, the computational cost of the key step, which is finding cycles, is constant level $\mathcal{O}(1)$.

However, when we set $loc > 1$, in the worst case, the algorithm has to consider all the vertices of unmarked strokes, since there is always a way of constructing the graph to push all the vertices in the graph have the distance at most 2 to the new vertex. A typical example is a star graph. In this case, the time complexity is $\mathcal{O}(|V| + |E|)$. Since there is no cycle in the previous graph, and the degree of the new vertex is at most 2, we have $\mathcal{O}(|V|) \sim \mathcal{O}(|E|)$. Thus the time complexity in the worst case is $\mathcal{O}(|V|)$, which is no worse than the state-of-the-art method [17].

Fortunately, the worst case mentioned above is an extreme case. In application, users' input strokes usually have high spatial locality and this bad event happens rarely according to our user study. In conclusion, our algorithm has only constant time complexity level $\mathcal{O}(1)$ when we set the highest local-sensitivity. And the time complexity would raise if we set a higher loc value. It would be $\mathcal{O}(|V|)$ in the worst case and it is decided by the local sensitivity, $i.e.$, loc, in normal cases. The results about time complexity in application can be found in the experiment part.

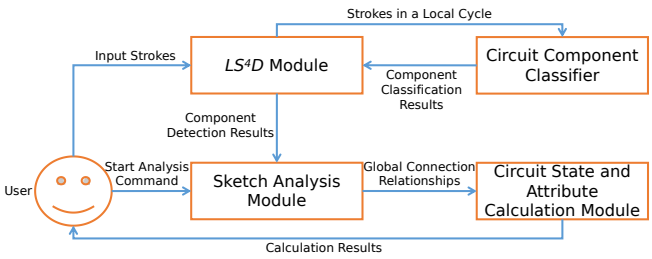


FIGURE 3. System modules and their relationships.

V. EXPERIMENTS AND USER STUDY

A. PLATFORM DESIGN

In order to show the efficiency of our circuit sketch symbol detection algorithm (LS^4D), we design a pen-based interactive circuit recognition and analysis system, on which we perform all the following experiments and comparisons. It consists of five modules: UI module, sketch analysis module, circuit state and attribute calculation module, and circuit component classifier. UI module interacts with users directly and takes users' input strokes. These strokes are processed by LS^4D module using the algorithm mentioned above to detect and localize sketch components. It relies on circuit component classifier, which is a convolutional neural network (CNN), to mark strokes with their belonging components. After finishing all strokes, user need to send a start-analysis command to notify the sketch analysis module. It utilizes detection results from LS^4D module to analyze the connection relationship between each components. Circuit state and attribute calculation module takes advantage of the results in sketch analysis module to make a list of equations about unknown attributes and solves it to get the results. The dependency relations of these modules are shown in Fig.3.

1) Interface Design

As shown in Fig.4, the sketch interface of our system is simple. There is no need for users to learn to use it, but just draw on it directly. The system would detect circuit components, and draw a rectangular box with a specific color around it. Different kinds of components are assigned different colors. The type of current drawn component is also shown in the left-top corner. When system detects a circuit component, a small button would appear near this component and there would be an input box if the user clicks on the button, which is used to input some attributes for it, $e.g.$, current, voltage, resistance values, etc. The input information would be shown in green. After user finishes sketch of the entire diagram, click on *Start Analysis* button to start calculating the remaining attributes and states of the diagram and calculated results would be shown near each component in red.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3035545, IEEE Access

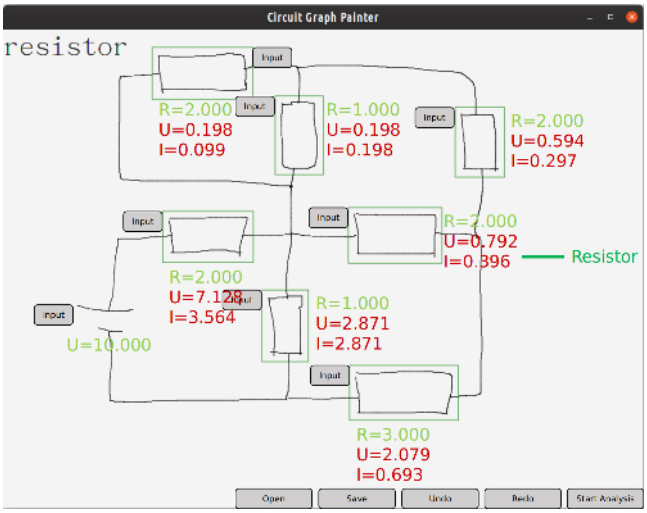


FIGURE 4. **Interface and running example of our prototype.**

2) LS^4D Module

LS^4D module is based on the main algorithm mentioned in Section III. Some low-level implementing details are provided here.

The sketch interface in our system is an electronic handwriting screen with 1920×1080 resolution. Similar to a down-sample process, we view the screen as a 640×360 checkerboard, where each cell is a 3×3 pixel area. And a 640×360 map is used to denote each cell is occupied by which stroke. Initially all the entries of this map are empty and an entry would be set as current stroke number if user's nib touches on a pixel belonging to this cell. This map is convenient for stroke collision detection, which is useful for updating our graph model and the down-sample process makes sure that this map does not consume too much memory.

On the other hand, in real cases, since sketch is inaccurate, stroke might not connected exactly with the endpoints being connected. In other words, there may be a subtle gap between two semantically connected points. Based on this observation, we relax the criterion of detecting inter-stroke collisions. As long as the distance between two endpoints is less than 5 cells, $i.e.$, 15 pixels, these two endpoints would be viewed as joint. This value is set based on most users' sketch habits in order to minimize the error rate from this aspect.

In addition, we employ the stroke segmentation approach in [21] besides our original segmentation method to relieve this problem, to handle cases that users draw multiple components with one stroke. At last, there are only the following restrictions or concerns to users' input:

- the hyper-parameter loc is set to 1 as default to reduce computational cost and users need to finish a component with a closed border in at most 4 strokes (a higher loc can relax this restriction and when $loc = inf$ this restriction would not exist);
- a distance less than 5 cells, $i.e.$, 15 pixels would be viewed as being connected. Users need to control the distance to make it equal to a value more than that when they want a pair of disconnected endpoints and less than that when they want a connected pair;
- a detail should be noted that when two wires cross with each other, the system would not view them being connected and users can divide one wire into two or put a dot on the crossing point to achieve connecting, which is consistent with common drawing standards.

Compared with previous arts like [2] [3] [4] [17], the methods and system in this work do not require users to present their strokes as a specific order, or maintain the whole screen clean at any time as shown in following user study.

3) Sketch Analysis Module

Detection for Other Symbols The LS^4D algorithm can detect those sketch symbols with closed border efficiently. However, in the scene of circuit sketch recognition, symbols of power supply and capacitor are not equipped with this characteristic. Thus, LS^4D fails to detect sketch of these symbols and we need some other mechanisms to tackle this problem. In this paper, we continue using our graph model to find these symbols.

As shown in Tab.2, symbols of power supply and capacitor consists of two unconnected strokes. In addition, strokes belonging to these components usually have only one wire connected. This is unlike those strokes belonging to the wires, since both ends of wires are connects by components in a normal circuit graph. In other words, while detecting these symbols, we can pay attention to vertices with degree 1, and verify their distances between each other as well as stroke lengths to decide whether they belong to power supply, capacitor, or just noise.

Connection Relationship Analysis With fine-tuned LS^4D algorithm and a simple mechanism to locate power supplies and capacitors, we can detect and locate all the circuit components as well as divide all the strokes according to the components they belong to. Now we can take advantage of these previous results to analyze their relationships between each other and convert user's input strokes into format needed by future calculation on the circuit diagram to implement our sketch analysis module. Firstly, all strokes remaining a -1 mark would be viewed as wires in circuit diagrams. Due to the flexibility of wires in the circuit graph, it is equivalent to directly use a stroke's corresponding vertex in our dual graph model to represent a node in the actual circuit graph. Therefore, edges between marked strokes and strokes with a -1 mark describe relationships between circuit components and wires perfectly.

4) Circuit State and Attribute Calculation Module

This part introduces the circuit state and attribute calculation module of our system. After extracting the connect relationships between all the wires and components, the last step is to calculate the state and attribute of the circuit graph. The basic idea is to construct a set of equations according to electrical

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3035545, IEEE Access

Component Type	Total # of Samples	Test Accuracy
Light Bulb	174	90.9%
Resistor	205	98.0%
Ammeter	184	92.9%
Voltmeter	181	92.3%
Diode	228	94.6%
Transistor	242	91.4%
Motor	236	92.2%
Buzzer	235	95.2%

TABLE 3. Details of our data set and test results

laws like Kirchhoff's circuit laws [22] and Ohm's law [23]. Specifically, we can use nodes converted from strokes with -1 (wire) mark to build equations for electric current with Kirchhoff's current law, use cycles from positive pole of power supply to negative pole to build equations for voltage with Kirchhoff's voltage law, and use detected circuit components to build equations with Ohm's law. Due to the forms of these equations, the set of equations constructed above can be converted into a linear system of equation. Therefore, we can consider using Gauss elimination method [24] to solve it. As long as conditions are sufficient, the rank of the matrix is no less than the number of unknown quantities and the solver can find the solution. The details of the solving process is not the core aspects of this paper.

5) Circuit Component Classifier

Sample Source : The training and test samples are collected from 11 users, including on-the-job and retired teachers, college students, and middle and high school students, so that the final model is oriented to people with different sketching habits, and overall classification accuracy can be as high as possible. Altogether, we collect about 200 hand-drawn samples for each type of circuit component with a closed border, and the total number of samples is over 1600. The details of our data set are shown in Tab.3. We randomly divide all samples of each type of component into a training set, a cross-validation set, and a test set according to the $7:2:1$ ratio.

Data Preprocessing : In order to consider a variety of different sketch styles as well as the equivalence of horizontal and vertical drawing of some components, we perform data augmentation on the training samples [25], as follows:

- for diodes, transistors, light bulbs, resistors, and buzzers, carry out the horizontal and vertical symmetry transformations on the images, as well as 90, 180, and 270 degrees rotation transformations;
- for diodes, buzzers, and resistors, carry out 1.2, 1.5, and 1.8 times compression transformations horizontally and vertically;
- for motors, ammeters and voltmeters, carry out the horizontal symmetry transformation.

The final training set consists of all original images and transformed images, with over 6500 samples in total.

Training and Testing Details : We use LeNet-5 [26] as our basic network architectures. Different from the original network, we use dropout with a probability of 0.5 for the first two fully connected layers to avoid overfitting [27] and replace original ReLU activation functions with LeakyReLU [28]. We adopt Adam [29] ($\alpha = 10^{-3}$,$\beta_1 = 0.9$,$\beta_2 = 0.99$) as optimizer, with a batch normalization strategy (batch size is 2). After each epoch, the model is tested on the cross-validation set. In the end, our model achieves a 99.9% accuracy rate on the training set, a 99.7% accuracy rate on the cross-validation set, and an overall accuracy rate of 93.7% on the test set after the entire training. The accuracy rates of each type of component are shown in Tab.3.

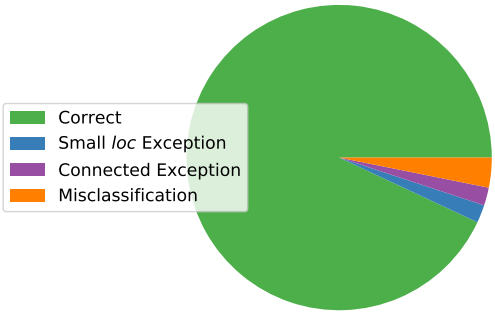


FIGURE 5. Results in user test for recognition accuracy.

B. USER STUDY

1) Recognition Accuracy Test

We convey user study on 17 users to test our algorithms and system. Each user draws 6 circuit diagrams provided by us from official tests for high school students (part of users also drew some other samples designed by themselves). The result is shown in Fig.5. A hand-drawn diagram is viewed as a successfully analyzed sample only if no error or exception occurs in all the modules. There are altogether 158 samples collected by us, with 147 successful analyses and 11 anomalous ones. Therefore, the overall analysis accuracy is 93.04% and all the exceptions fall in the range of drawing restrictions mentioned above. Note that users were not informed about these restrictions before the test. Among these 158 samples, we observe that 27 cases containing interspersed drawn symbols, which cannot be handled by approaches like [3], [2], [15], etc. Numerically, the average accuracy is also higher than previous state-of-the-art approaches in [3], [17] and [4].

For specific instances, it turns out that our algorithm and system can not only face regular circuit examples from physics text books, but also handle messy input with some noises and complex problems, as shown in Fig.6 and Fig.7. Here we hide the input interface to make a clean panel.

There are three types of exceptions. The first one is due to default loc value. When $loc = 1$, theoretically LS^4D can only find cycles with length at most 3 if the rectangular closure is not introduced and if users draw a component with strokes more than that number, there could be a small loc exception. Fortunately, this can be solved by using a higher loc value and sacrifice some running efficiency. The second is about the setting of connecting pixel error tolerance

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3035545, IEEE Access

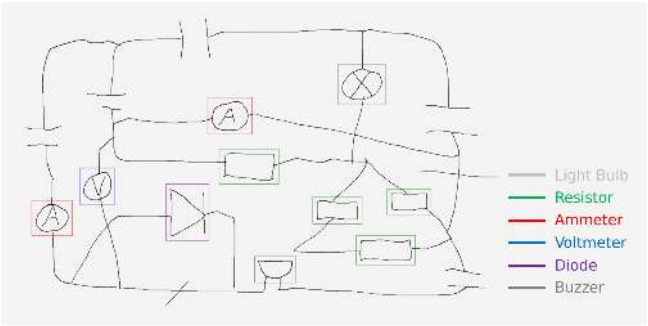


FIGURE 6. **The system is robust enough to handle messy and irregular sketch with mistakenly touching noises.**

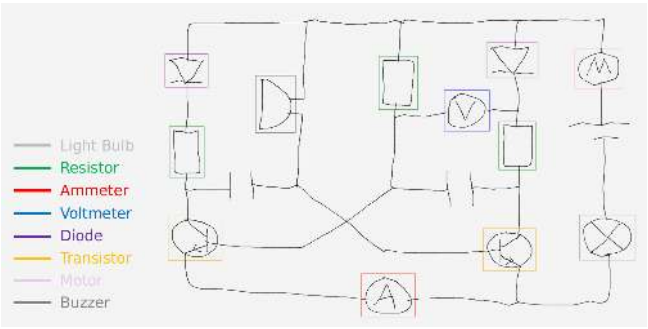


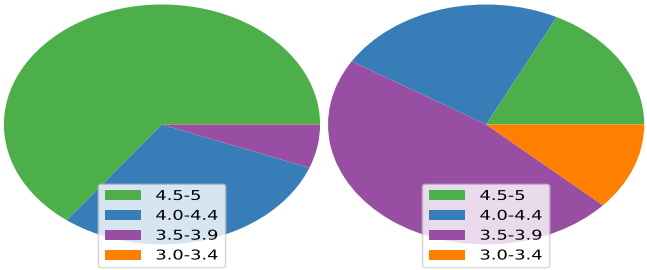
FIGURE 7. **The algorithm can work on complex and sophisticated problems efficiently.**

mentioned above. We set a 15-pixel threshold to decide whether two strokes are connected. In these exception cases, users represented disconnecting cases with a distance smaller than 15 pixels or represented connecting ones with a distance higher than that value. Setting this value more reasonably is a meaningful research topic that is beyond the scope of this paper. And the last type is about misclassification from the classifier.

2) User Satisfaction

We also carry out a satisfaction evaluation based on a 5-point Likert scale in contexts of circuit teaching and learning, comparing with a GUI-based commercial circuit simulator [1], which requires users to drag icons and connect components by mouse. The rating results are shown in Fig.8. Since pen-based interaction is more natural, efficient, and vivid for students and teachers in class, most rating results for our prototype concentrate on level $4.5 \sim 5.0$, which are better than results using mouse-interaction technology. Some representative comments on prototype based on our approach are listed below:

- I think the pen-based system has higher practical value and it covers all points of electricity in middle and high school. I am satisfied with the accuracy and efficiency of this recognition and analysis system, and look forward to more helpful functions in teaching such as highlighting and blackboard-writing (from a physics teacher);



(a) Pen-Interaction. (b) Mouse-Interaction

FIGURE 8. **Rating from all tested users for two kinds of prototypes.**

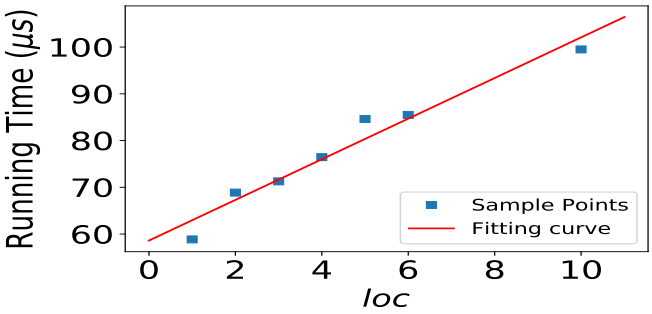


FIGURE 9. **Relationship between running time and loc.**

- The most valuable use of this system for me is to check my homework answers efficiently because it supports sketch and I can draw circuits freely on it as convenient as drawing on a scratch paper. It can also help me calculate solutions in real time and develop my understanding on electrical laws. Thanks to this system, I am no longer afraid of circuit diagram problems. (from a high school student).

C. RUNNING TIME EXPERIMENT

In this part, we use different loc values to evaluate the average per-stroke running time of our LS^4D algorithm using a uniform standard circuit shown in Fig.4. We sample the setting of loc from $1 \sim 6$ and 10. The corresponding results are shown in Fig.9. Under default loc value 1, we get a latency less than $60\mu s$ per stroke on average on a modern workstation with an Intel-9980HK CPU. To a general trend, the running time and loc show a linear relationship. The results indicate that the key steps in our algorithm only take roughly $100\mu s$ even though loc is as high as 10 and users can hardly sense the system delay. As a result, 100% of users are satisfied with real-time performance of our system.

VI. CONCLUSION AND FUTURE WORKS

This paper proposes a method to model user's drawn strokes with vertices in graphs, and an efficient and reliable locality-sensitive algorithm to detect and localize symbols that have closed borders through information of cycles. The algorithm has solid theoretical foundation. Based on it, the circuit recognition and analysis system produces 93.04% accuracy on a user experiment with 158 samples. In this process, we

publish the source code of our algorithm, model, and data set used by sketch circuit component classifier, which facilitates researches in the field of sketch recognition, especially for circuit sketch recognition.

Since general LS^4D algorithm works for all symbols with a closed border, it can also be applied to other tasks related to sketches, like recognition of sketches of flow charts and UML diagrams. In the future, we would also like to extend the system to a wider range of applications, $e.g.$, to build it as a real educational software that can be deployed in classrooms, to apply the algorithm into systems of flow chart and UML diagram recognition and analysis, and to explore the value of the algorithm in some CAD tasks like sketch coloring.

REFERENCES

[1] "Digikey Electronics," https://www.diikey.com/schemeit/project/, 2020.
[2] L. Gennari, L. B. Kara, T. F. Stahovich, and K. Shimada, "Combining geometry and domain knowledge to interpret hand-drawn diagrams," Computers & Graphics, vol. 29, no. 4, pp. 547–562, 2005.
[3] C. Alvarado and R. Davis, "Dynamically constructed bayes nets for multi-domain sketch understanding," in ACM SIGGRAPH 2007 courses, 2007, pp. 33–es.
[4] T. M. Sezgin and R. Davis, "Sketch recognition in interspersed drawings using time-based graphical models," Computers & Graphics, vol. 32, no. 5, pp. 500–510, 2008.
[5] O. Altun and O. Nooruldeen, "Sketrack: Stroke-based recognition of online hand-drawn sketches of arrow-connected diagrams and digital logic circuit diagrams," Scientific Programming, vol. 2019, 2019.
[6] P. Hu and D. Ramanan, "Finding tiny faces," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 951–959.
[7] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," arXiv preprint arXiv:1804.02767, 2018.
[8] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 3431–3440.
[9] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2014, pp. 580–587.
[10] B. Edwards and V. Chandran, "Machine recognition of hand-drawn circuit diagrams," in 2000 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No. 00CH37100), vol. 6. IEEE, 2000, pp. 3618–3621.
[11] M. D. Patare and M. S. Joshi, "Hand-drawn digital logic circuit component recognition using svm," International Journal of Computer Applications, vol. 143, no. 3, pp. 24–28, 2016.
[12] Y. Liu and Y. Xiao, "Circuit sketch recognition," Department of Electrical Engineering Stanford University Stanford, CA, 2013.
[13] R. De Silva, D. T. Bischel, W. Lee, E. J. Peterson, R. C. Calfee, and T. F. Stahovich, "Kirchhoff's pen: a pen-based circuit analysis tutor," in Proceedings of the 4th Eurographics workshop on Sketch-based interfaces and modeling, 2007, pp. 75–82.
[14] M. J. Fonseca, C. Pimentel, and J. A. Jorge, "Cali: An online scribble recognizer for calligraphic interfaces," in AAAI spring symposium on sketch understanding, 2002, pp. 51–58.
[15] M. Liwicki and L. Knipping, "Recognizing and simulating sketched logic circuits," in International Conference on Knowledge-Based and Intelligent Information and Engineering Systems. Springer, 2005, pp. 588–594.
[16] J.-P. Valois, M. Côté, and M. Cheriet, "Online recognition of sketched electrical diagrams," in Proceedings of Sixth International Conference on Document Analysis and Recognition. IEEE, 2001, pp. 460–464.
[17] G. Feng, C. Viard-Gaudin, and Z. Sun, "On-line hand-drawn electric circuit diagram recognition using 2d dynamic programming," Pattern Recognition, vol. 42, no. 12, pp. 3215–3223, 2009.
[18] J. F. Dreijer, "Interactive recognition of hand-drawn circuit diagrams," Ph.D. dissertation, Stellenbosch: University of Stellenbosch, 2006.
[19] "circuit-symbols-poster," https://www.philipharris.co.uk/product/lab-equipment/lab-essentials/tools-and-sundries/circuit-symbols-poster/b8r06517, 2019.
[20] "Bridge (graph theory)," https://en.wikipedia.org/wiki/Bridge_(graph_theory), 2019.
[21] G. Feng, "Hmm-based stroke fragmentation"," Technical Report, Ecole Polytechnique de lâĂŹUniversité de Nantes, Tech. Rep., 2007.
[22] "Kirchhoff's circuit laws," https://en.wikipedia.org/wiki/Kirchhoff's_circuit_laws, 2020.
[23] "Ohm's law," https://en.wikipedia.org/wiki/Ohm's_law, 2020.
[24] S. Parter, "The use of linear graphs in gauss elimination," SIAM review, vol. 3, no. 2, pp. 119–130, 1961.
[25] L. Perez and J. Wang, "The effectiveness of data augmentation in image classification using deep learning," arXiv preprint arXiv:1712.04621, 2017.
[26] R. Al-Jawfi, "Handwriting arabic character recognition lenet using neural network." Int. Arab J. Inf. Technol., vol. 6, no. 3, pp. 304–309, 2009.
[27] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," The journal of machine learning research, vol. 15, no. 1, pp. 1929–1958, 2014.
[28] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," arXiv preprint arXiv:1505.00853, 2015.
[29] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.

SONGHUA LIU enrolled in the Department of Computer Science and Technology, Nanjing University, Jiangsu, China in 2017. He is also a student in the State Key Lab. for Novel Software Technology. At present, he is an undergraduate and pursuing the BS degree. His research interests include artificial intelligence in human-computer interaction, computer-aid artistic creation, and federated learning.

LINFENG LI enrolled in the Department of Software Engineering, Nanjing University, Jiangsu, China in 2017. At present, he is a junior student. His research interests include artificial intelligence in autopilot and human-computer interaction.

FUKANG JIU enrolled in the Department of Computer Science and Technology, Nanjing University, Jiangsu, China in 2017. At present, he is a junior student. His research interests include knowledge graph and artificial intelligence in human-computer interaction.

GUIHUAN FENG received the B.S. degree and the Ph.D. degree in computer science and technology from Nanjing University, Nanjing, China, in 2003 and 2009, respectively. In 2003, she joined the State Key Laboratory for Novel Software Technology, Nanjing University. She is currently an Associate Professor with the Software Institute, Nanjing University. Her research interests include intelligent human-computer interaction, sketch-based interaction and pattern recognition.