

## Sketic: a machine learning-based digital circuit recognition platform

Mohammad ABDEL-MAJEED\*, Tasneem ALMOUSA, Maysaa ALSALMAN, Abeer YOSF

Department of Computer Engineering, University of Jordan, Amman, Jordan

Received: 05.10.2019

Accepted/Published Online: 04.02.2020

Final Version: 29.07.2020

**Abstract:** In digital system design, digital logic circuit diagrams are built using interconnects and symbolic representations of the basic logic gates. Constructing such diagrams using free sketches is the first step in the design process. After that the circuit schematic or code has to be generated before being able to simulate the design. While most of the mentioned steps are automated using design automation tools, drafting the schematic circuit and then converting it into a valid format that can be simulated are still done manually due to the lack of robust tools that can recognize the free sketches and incorporate them into end user simulators. Hence, the goal of this paper is to construct and deploy computer simulation tools capable of understanding free sketches and incorporate them into useful simulation tools. Such a tool will be useful at both the educational and the industrial levels. Moreover, while this tool is designed to deal with sketched logic circuits, it can be generalized and applied to many other fields to convert the sketched design into a digital format. To implement this tool, we relied on the emerging machine learning and image processing concepts to make sure that the designed system is robust and accurate. Our results show that our system is able to recognize all the gates in the digital circuit with more than 95% accuracy.

**Key words:** Digital logic circuits, logic gates, machine learning, deep neural networks, detection, recognition, classification

### 1. Introduction

In digital system design, digital logic circuit diagrams are built using interconnects and symbolic representations of the basic logic gates. Constructing such diagrams is the first step in the design process. Free or pen-and-paper sketches are usually used in the early stages of constructing the circuit diagram to convey concepts and ideas. After the design is completed it is incorporated into useful digital logic circuit simulators. Many formats can be used to incorporate the digital logic circuit. First, the schematic of the circuit can be drawn using drag and drop environments [1]. Second, the circuit can be implemented using hardware description languages (HDLs) [2]. Then the designed schematic or the written code will be used in the simulation environment to test the operation and functionality of the designed circuit.

While most of the above-mentioned steps are automated using design automation tools, drafting the digital circuit and then converting it to a valid format are not automated efficiently. Several works have been proposed to automate this step [3–7]. The work proposed by Zamora and Eyjólfssdóttir [7] relies on the pen strokes on the computer or tablet screen to recognize the drawn digital circuit. Based on the strokes the software will be able to detect the gates. This process will be done concurrently with the drafting step. Adapting such tools will depend on their ability to recognize circuit elements accurately. While such tools appear sufficient

\*Correspondence: m.abdel-majeed@ju.edu.jo

we think that circuit recognition is a challenging task because of the high variation in the drawing styles from one person to another and the performance of the tools will rely on the drawing style of the user. In addition, the sketching process may require erasing and correcting sketches before finalizing them. Hence it is more convenient to perform the recognition step after the end of the sketching step. To make such a process easier and more comfortable, in the present paper we propose to separate the drafting step and the circuit recognition step to give the user more freedom during the drafting step and at the same time make the recognition step more efficient and easier.

Hence, the goal of the present paper was to construct and deploy software algorithms that are capable of understanding free sketches and incorporate them into useful tools accurately. Such a tool will be useful at both the educational and the industrial levels. At the educational level, such a tool can be used to help university students at early stages to incorporate their circuits quickly into useful simulation tools to easily and quickly apply and verify the learned concepts. It can also help the instructor to validate the digital circuits drawn by students during exams and homework efficiently. At the industrial level, such a tool will play a major role in saving the time and effort of design engineers by automating the process of converting the sketches into a ready-to-simulate design. Hence, they can put more effort in coming up with a good design.

Adapting such a tool will depend on its ability to robustly convert the free sketch. This includes the ability to recognize all the components in the circuit and relate them to each other accurately. However, circuit recognition is a challenging task because of the high variation in the drawing styles from one person to another. For example, if we ask students to draw an OR gate we will find that the collected samples will vary in the curve style, the size of the gate, the direction of the gate, and so on. To solve these issues in the present paper we relied on the emerging machine learning [8–11] as well as image processing algorithms to come up with a robust algorithm that can accurately detect and recognize all the components in the free sketch. Hence, in the present paper we propose Sketic. Sketic is a robust machine learning-based digital circuit recognition platform. It is designed to take a digital circuit sketch image as an input and generate the Verilog code for the sketched circuit as an output.

While in the present paper we focused on digital logic circuits, the proposed techniques can be extended to any type of sketched circuits or structures with predefined basic blocks. RLC circuits and transistor level designs like amplifiers and filters are some examples of circuits that can be detected and recognized by our proposed approach with minimum effort.

In the next sections we will go over the details of the design of Sketic and how it handled the variations in the drawing style and noise. The paper is organized as follows: in Section 2 we will talk about the related works. In Sections 3 and 4 we will go over system design and implementation. We give our conclusions in Section 5.

## 2. Related works

Computer-aided design (CAD) applications like Logisim [4] and Modelsim [1] are widely used to help you design, simulate, and debug logic circuits. These applications require the user to manually enter the desired circuit either as a schematic or a hardware description language (HDL) code. Most designers find the latter step tedious and inefficient, especially as the initial circuit draft is usually completed using pen and paper.

There are several research efforts in the domain of recognizing digital circuit sketches. The proposed techniques rely either on the recorded strokes during the sketching process (i.e. online recognition) or on applying image processing and classification algorithms on an image of the hand-drawn circuit (i.e. offline recognition). Next, we discuss in detail the related work in each class.

**Online recognition:** Multiple research papers have focused on recognizing digital circuits sketches based on the pen strokes while drawing the circuit draft [3, 5–7, 12]. The SketchRead [5] system uses shape description language to describe shapes. During the sketching process, shapes are recognized by analyzing the strokes and finding possible interpretations. CircuitBoard [7] is another online recognition engine that identifies the logic circuits sketches and incorporates them into circuit simulators automatically. In CircuitBoard, during the sketching process the strokes are recorded and classified into linear, curved, or adjacent colinear lines. As soon as the user releases the pen, the drawn part is sent to the recognition engine, which relies on the strokes' patterns and the recorded list of events to interpret the drawn gate. LogiSketch [6] uses algorithms like Rubine, which relies on the sequence of strokes to recognize the drawn shape. Most recently, DCSR [12] uses the \$P\$ algorithm to record the strokes needed in the recognition step.

**Offline recognition:** Instead of relying on the strokes, multiple research papers have proposed starting the recognition after the circuit draft is complete. In these papers, image processing techniques followed by classification and recognition algorithms are applied [13, 14]. Patare and Joshi [13] leveraged image processing algorithms like region-based segmentation to segment the image of the sketched digital circuit and then used the support vector machine (SVM) algorithm to recognize the gate in each segment. The feature vector fed to the SVM algorithm is the Fourier descriptor of the segment. Likewise, Datta et al. [14] applied image segmentation techniques to segment the circuit image and generate the feature vector, which is consequently fed to the decision tree (DT) algorithm for recognition.

The research efforts in this domain are not limited to digital circuits but they have been extended to recognize electrical circuits [15–17]. The authors in previous studies [15, 16] applied image segmentation and feature extraction of electrical circuit elements (e.g., resistor and capacitor). The extracted features included the histogram of oriented gradients, local binary pattern (LBP), statistical features based on pixel density like scalar pixel-distribution, and vector relationships between straight lines in polygonal representations. The SVM algorithm is applied to the extracted features to classify each segment into the correct electrical circuit element. Dewangan and Dhole [17] used the shape area, major axis length, minor axis length, centroid, orientation, eccentricity, and convex area to generate the feature vector and then applied the K-nearest neighbor (KNN) algorithm to classify the circuit elements into their correct class.

The Sketic technique proposed in the present paper is an offline recognition platform for digital circuits; it makes the recognition more efficient and conformable because it does not rely on the sequence of strokes as in previous studies [3, 5–7, 12]. Instead, the input to Sketic is an image of the complete drafted circuit that can be simply taken using a mobile camera. Sketic leverages image processing and machine learning algorithms to recognize all the components in the circuit (i.e. logic gates, wires, inputs, and outputs), relate them to each other, and generate the Verilog code that implements the circuit. Thus, Sketic automates the design entry step and makes it much more time efficient by converting the hand-drawn circuits' drafts into Verilog codes.

The SVM, DT, and KNN machine learning algorithms used in the previously proposed techniques require careful and accurate feature selection and extraction in order to achieve acceptable recognition results. However, manually extracting the features as done previously [13–17] requires deep analysis of target elements. In addition, for each new digital block or circuit element the feature extraction procedure might be repeated with new features required to resolve conflicts and similarities. To address these drawbacks, Sketic relies on deep neural networks (DNNs) and convolutional neural networks (CNNs) to implicitly extract the features of the logic gates during the training phase. This improves the scalability of Sketic by allowing it to support new logic

blocks, inputs, and outputs seamlessly. More importantly and as will be shown in the results section, the use of DNN and CNN machine learning algorithms causes the recognition accuracy of Sketic to reach 99%, which is higher than what was reported in previous studies [13, 17]

### 3. System design and implementation

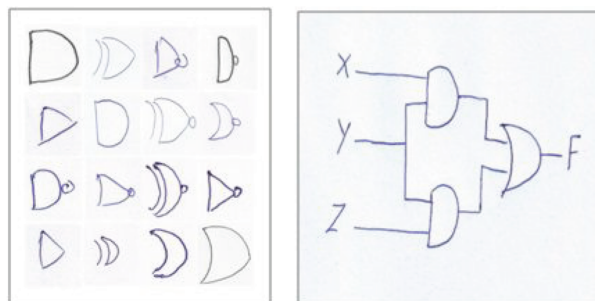
As mentioned before Sketic should be able to detect and recognize the sketched circuit accurately. To achieve this goal Sketic should be able to handle the wide variation in the sketching styles with minimum constraints; distinguish between wires, gates, inputs, and outputs; know the type of the gate and recognize its input and output names; and relate the identified circuit components to each other to generate the final design description in Verilog accurately. Accordingly and to be able to deal with these challenges the following stages are the building blocks of Sketic:

1. Sample collection and preprocessing: Collect a large and wide range of samples for logic gates and circuits that will be used in supporting the design of the other stages.
2. Detection and recognition stage: Responsible for detecting the circuit components and classifying them into their specific class like gates (AND, OR, INV, NAND, NOR, XOR, and XNOR), letters, and wires.
3. Mapping stage: Maps connected components in the circuit and relates them to each other.
4. Code generation stage: Generates the Verilog code of the circuit.

In the following subsections we will cover the stages in details.

#### 3.1. Sample collection and preprocessing

At the end Sketic should be able to recognize circuit elements like gates wires and letters. To be able to do that, Sketic should be trained on circuits that contain all these elements to be able to achieve the final goal. Hence a large number of samples of logic gates and logic circuits were collected from university students who study computer engineering. Students were asked to draw sketches of the basic gates separately (AND, OR, INV, NAND, NOR, XOR, XNOR, and buffer) and digital circuits that implement certain functions. Figure 1 shows examples of the collected samples.



**Figure 1.** Examples of the collected samples.

The character samples were adapted from “NIST Special Database 19” [18]. NIST publishes hand-printed sample forms from 3600 writers and 810,000 character images isolated from their forms. Sketic is trained on the most commonly used letters in logic diagrams. It is trained on upper case A, B, C, D, F, W, X, Y, Z.

The purpose of the preprocessing step is to filter the sample images and generate the feature vector that will be used in the training phase. The preprocessing is important because the sample images may have some variations in size and colors. These variations are due to the variations in the drawing styles and the camera used to take photos of the samples. Thus we applied noise filters (like the median filter) to the collected samples, unified the size of the images to  $50 \times 50$  pixels, and applied the close operation for all the objects in the image. After this stage the collected samples will be the reference to measure the efficiency of the design of the subsequent stages that are discussed below.

### 3.2. Detection and recognition stage

The detection and recognition stage is responsible for detecting the circuit components and classifying them into their specific class like gates (AND, OR, INV, NAND, NOR, XOR, and XNOR), letters, and wires. Several detection and recognition algorithms can be used. Using image processing functions to implement the detection and recognition stage is complicated and will be subject to the variations caused in the drawing styles. Moreover, relying solely on the image processing functions may make the scalability and the possibility of expanding the system complicated and time consuming. Hence in the present paper we will explore using two machine learning algorithms to perform the detection and recognition task. We will explore using NNs and CNNs.

#### 3.2.1. Detection and recognition stage using neural networks

In this subsection we will go over the details of implementing Sketic using NNs. At the detection level we will discuss the letters and gates detection procedure. At the recognition level we will discuss the classification of the detected elements into their specific classes.

##### 3.2.1.1. Detection stage

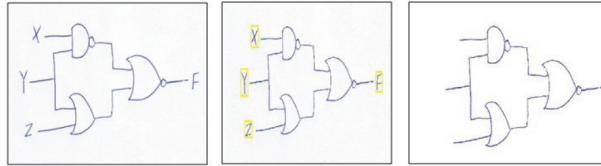
In order for the NN to be able to recognize circuit components, the first step in this stage will be separating the connected circuit components like wires, gates, inputs, and outputs. Hence, image segmentation can be used to split the digital circuit components. However, the amount of constraints placed on the detection algorithm defines its complexity. For example, if we force the user to draw wires in a different color than the gates or avoid gates that can cause confusion like XOR then the segmentation procedure will be easier. However, in our system design we avoided such constraints to make sure that our system works for a wide range of digital circuits.

The basic algorithm starts by performing the close operation for open objects (gates that are not closed properly), filling the holes in the full circuit, removing the wires by performing the open operation, and subtracting the filled image from the original image, which will result in an image that only contains the gates. Finally a bounding box is created around the basic objects.

Using these steps, we are able to locate the gates in the image. However, there are three reasons that make this basic algorithm not perfect for our goals. First, the method could not get the letters (i.e. x, y, z, and f) that are important for generating the Verilog code. Second, this technique is not able to detect the complete XOR gate and it will detect only the OR part because the arc part of the XOR gates is not connected to the other part of the gate. Third, sometimes wires intersect with each other and create a loop that can be confused with a gate.

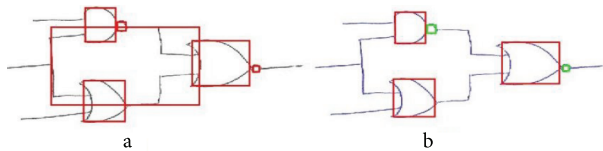
Taking these deficiencies into consideration we modified the algorithm to handle them. In the modified algorithm we will detect the letters first, then the gates, and then the wires. In the following paragraphs we will go over the details of each detection algorithm used.

**Detecting letters:** Sketic starts by locating letters in the circuit. After analyzing a wide range of sample digital circuits we found that letters representing inputs and outputs are located as separate objects that are not connected to wires or gates. Hence we start detecting the inputs and outputs using the bwboundaries function in MATLAB [19]. Bwboundaries places rectangles around the objects it finds and this includes inputs, outputs, and gates in our design. Since gates are going to be connected to wires (input and output wires) the rectangle representing the gate is going to be surrounded by its connected components and will not be wrongly classified as a letter. Figure 2 shows an example of letter detection of a circuit. After this step the letters will be removed from the circuit.



**Figure 2.** Illustration of letter detection.

**Detecting gates and bubbles:** After removing the letters from the circuit, bwboundaries is used to detect closed elements in the circuit. The closed elements can be gates, bubbles, or wire loops. At this step Sketic should extract gates and bubbles and ignore loops. To achieve this goal, we trace the internal boundaries of the components in the circuit using bwboundaries and draw rectangles around them as in Figure 3a. In our implementation and based on our analysis for a wide range of circuits we assumed that if an element is a normal loop then its rectangle is going to intersect with at least 3 other rectangles. If this is the case then the rectangle will be ignored. As a result the loop in Figure 3b is ignored. Then all rectangles are compared to each other to distinguish gate from bubble. In our implementation if the area of the rectangle normalized to the area of the largest rectangle is less than a certain experimental threshold (0.25 based on our analysis of a large number of samples) then we assume that the smaller rectangle represents a bubble. If it is larger than the threshold then it will be considered a gate. In Figure 3b gates are surrounded by red rectangles and bubbles by green ones.



**Figure 3.** Illustration of gate and bubble detection.

Our analysis of the sample gates showed that sometimes the applied algorithm does not work and should be modified to deal with some special scenarios. Single input loops require special handling. Single input loops are loops formed when the same input is connected to the two inputs of the gate. Such loops should be ignored because they are not part of the gate itself. Hence, in our implementation we assumed that if the smaller rectangle of the two neighboring rectangles is not a bubble (its size is higher than the threshold) then it is considered a loop and should be ignored.

The applied heuristics in this stage are experimental and are based on the hundreds of samples that we have. In the results section we will show that the applied heuristics are sufficient. However, having such a large number of heuristics may affect the accuracy, flexibility, and efficiency of the systems in certain scenarios.

**Detecting wires:** After gates and letters are detected, they are removed from the circuit, keeping only the wires. After the wires image is acquired it is cleaned to remove any unwanted components.

### 3.2.1.2. Recognition stage

After detecting all the components in the logic circuit we need to classify them into their types. For example, the gates should be classified into NOT, AND, OR, NAND, NOR, XOR, NNOR, etc. and the letters should be classified into their specific names like X, Y, Z.

To be able to recognize the gates and letters we should find a set of unique features for each one. Relying on the manually extracted image features for recognition may not be sufficient and requires a lot of work. Hence, in the recognition stage we relied on the emerging machine learning algorithms to perform the classification task quickly and accurately. Supervised learning algorithms like DT and SVM can be used to perform the task of classification of the gates and letters into their correct type. To achieve this the samples for each gate and letter should be analyzed to generate the features vector. The features can be something like symmetry, number of holes, and so on. The feature vector creation procedure should be done for each gate and letter separately. Such a procedure requires time and effort in selecting the correct features that describe each object accurately. Moreover, adding a new object (gate or letter) to the set of objects is not trivial and requires generating its feature vector and sometimes adding other features to distinguish it from other objects. For example, in our implementation we started with Logistic Regression One vs. All Method [20]. We started by training Sketic on the basic three gates, AND, OR, and INV. The accuracy of this model is 95%. However, as we started adding more gates, the accuracy started dropping and the training time started to increase.

To avoid such drawbacks in our implementation we used an artificial neural network (ANN) [21] to do the classification. The ANN generates the features for the classified objects automatically after a few iterations of forward and back propagation to the ANN, hence reducing the amount of effort and analysis that should be devoted to extract the features for each object (gate or letter in our system).

Hence we decided to switch to the ANN. The ANN, as any supervised learning algorithm, works in two steps. First it is trained on a large number of samples of the labeled data to generate the features for each object that should be recognized. For example, we feed to the ANN images of an AND gate and we clearly specify that the object in the image is an AND gate. At the end of the training the NN will form an accurate model that describes the AND gate. We do the same for all the objects (gates and letters) that should be classified. Second, after the training stage is done we feed to the ANN an image of an object without specifying what is in the image. The ANN should be able to accurately classify the image content into the correct class (i.e. gate type or input name).

ANN design has many parameters that can affect the performance of the NN. The number of hidden layers and the number of nodes in each layer are examples of such parameters. Since we want to find the network architecture that delivers the best accuracy, we need to do a search for the best parameters for the NN. Hence the data are divided into train, cross validation (CV), and test sets and sensitivity analysis for the NN parameters like the number of layers and nodes per layer is performed.

The designed system should be able to recognize letters, gates, and their direction. To achieve this we trained three different neural networks, one for each class (i.e. letter, gate type, and direction). Of course we can use one NN for all the classes instead of 3. However, we noticed that using a classifier for each certain class is better in terms of accuracy, timing, and scalability. Below are the details for each NN classifier:

1. Letters classifier Each detected letter is passed into the letters classifier that is implemented using the NN to get each letter's label.
2. Direction classifier This classifier is a NN classifier that labels each gate with its direction (up, down, left, or right). The direction classifier is a NN classifier that is trained on the four directions (left, right, up, and down) of gates.
3. Gates classifier The gates classifier consists of multiple classifiers that work together to obtain the correct classification of the gates in the circuit. The gates classifier is divided into the following classifiers:
  1. Basic gates classifier (AND, OR, BUFFER)
  2. XOR classifier (XOR)
  3. Bubbles classifier (NAND, NOR, XNOR, Inverter)

The basic gates classifier is a NN classifier that recognizes the basic gates of AND, OR, and BUFFER in the circuit. Figure 4 shows two examples of basic gates classification; note that all gates here are labeled as basic gates. As shown in the figure below the XOR is initially classified as an OR gate.

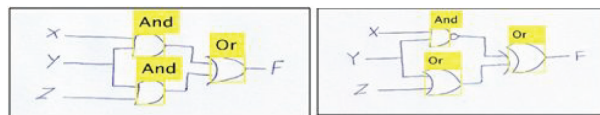


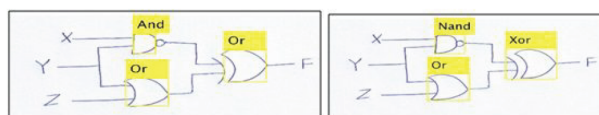
Figure 4. Basic gates classifier output.

Regarding the XOR gate, the gate is misclassified because of the additional arc that appears on the left of the gate. This arc is classified as a separate object and as a result will not be part of the right side of the gate. Hence the arc will be ignored and the right part of the gate will be classified as an OR gate. To solve this issue we implemented an additional classifier named the XOR classifier. The XOR classifier will only extend the boundary drawn around each recognized OR gate by 20% in the direction of the inputs (based on the result of the direction classifier discussed before). If the extended boundary box intersected with another boundary box then this means that there is an object that is very close to the gate. We assume that this object is the arc of the XOR gate and hence the gate is classified as an XOR. If there is no object then the gate is classified as an OR gate.

To solve the second issue of recognizing a NOR as an OR, NAND as AND, INVERTER as BUFFER, and XNOR as XOR we used an additional step named bubble classifier. This classifier maps each detected bubble to its gate and labels it with the inverted label. Here we will use the same procedure used in the XOR classifier. In the bubble classifier we will extend the boundary drawn around each recognized gate in the direction of the output. If the extended boundary intersects with a bubble object then the gate label is inverted (i.e. the AND will be labeled as NAND and so on). Figure 5 shows an example of a circuit before and after passing it to the bubble classifier.

**Classifiers discussion:** In our design we used three basic NN classifiers: gate, direction, and letter classifiers. The three classifiers are mainly multilayer NN classifiers. To make sure that each classifier gives us





**Figure 5.** Bubble and XOR recognition steps.

acceptable accuracy we ran it with different configurations. A summary of each classifier design is shown in Table 1.

The number of nodes in the output layers matches the number of classes. For example, in the direction classifier we classify the direction into one of four directions (up, down, right, left). Hence the number of nodes in the output layer will be 4 and so on. As shown, all the classifiers have accuracy that is higher than 96%.

**Table 1.** NN classifiers’ implementation details.

Classifier	Gates	Direction	Letter
Number of hidden layers	1	1	2
Number of nodes in input layer	2500	2500	2500
Number of nodes in hidden layer	5	4	49
Number of nodes in output layer	3	4	9
Accuracy	96.1%	99.7%	96.5%

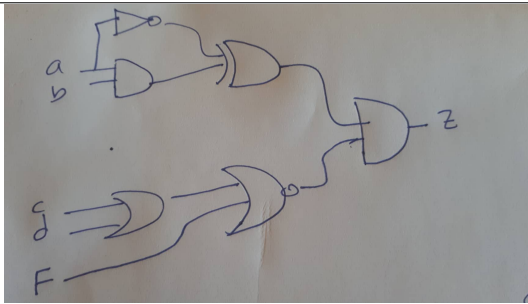
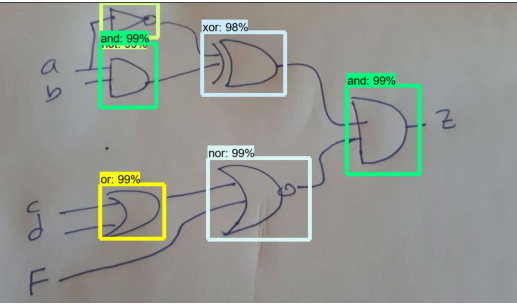
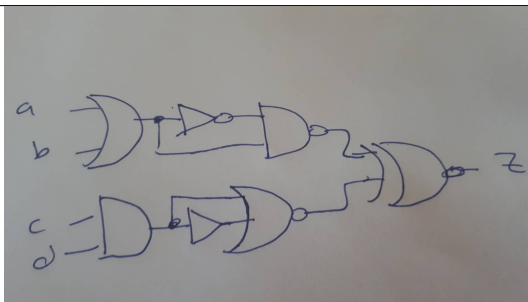
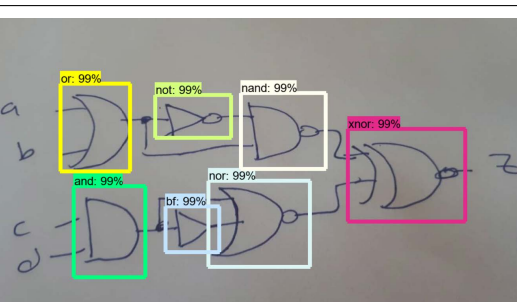
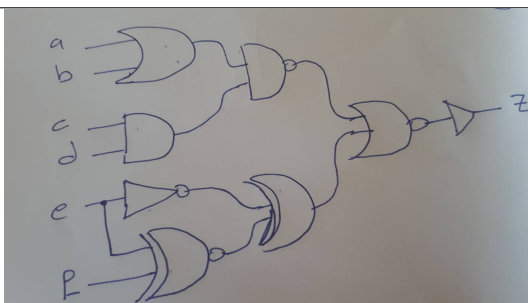
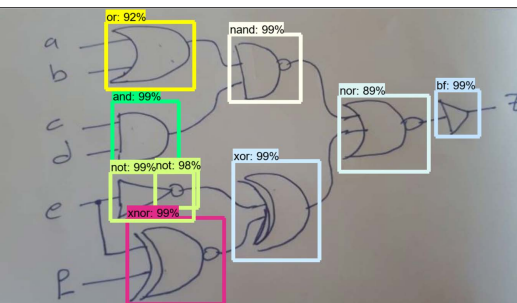
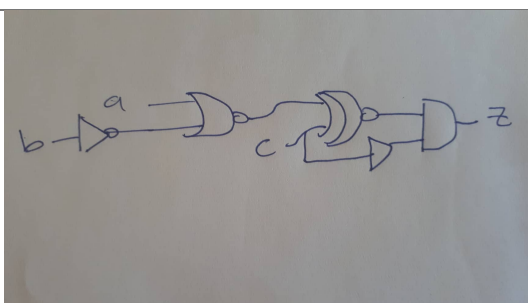
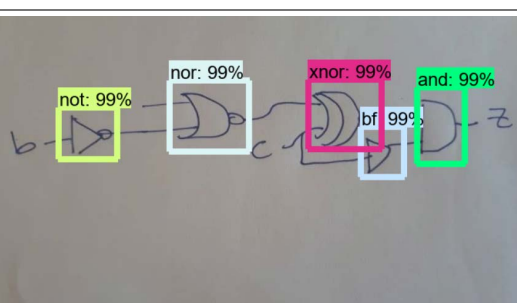
### 3.2.2. Detection and recognition using convolutional neural networks (CNNs)

A CNN is a modified NN that is designed to deal with images and videos. The CNN can be used to detect objects in an image [22]. Using a CNN instead of a NN can help us merge the detection and recognition steps into one step and simplify the implementation details when compared with the NN implementation discussed in the previous section as we will discuss later. In the CNN the system is trained the same way. It will be supplied with a large number of labeled images for each discrete gate and letters. The CNN will apply filters to the input image to detect the edges and the details of the objects. Making the CNN deeper (increase the number of layers) improves the ability to extract more fine grain details from the image and as a result the recognition will be more accurate. To be able to use the CNN to detect and recognize circuit elements the sliding window algorithm will be integrated with the CNN. The sliding window algorithm works by moving (sliding) a small window on the original image. The part of the image that falls within the sliding window will be fed to the CNN for recognition. The stride and the size of the window will decide the number of images that will be generated by the sliding window algorithm. Each generated image will be fed to the CNN to recognize the object inside the image (gate type and letter value). At the end the windows with above 50% prediction probability will be considered and their coordinates on the original image will be highlighted with the prediction result. One round of sliding window algorithm will cover all the original image.

Several algorithms have been proposed to enable object detection and recognition using CNNs. Yolo [23], SSD [24], and Faster-R-CNN [25] are the most recent algorithms used for object detection and recognition. We tried all three algorithms and decided to use Faster-R-CNN since it provides the best trade-offs between accuracy and speed for our application. Faster-R-CNN works by detecting the regions of the gates in the full circuit and then classifying the gates into their types directly. We will provide more details regarding the accuracy of these algorithms in the results section.

Table 2 shows samples of the logic circuits with different levels of complexity in the first column and the result of feeding these images to Sketic while using Faster-R-CNN as a detection and recognition algorithm. As shown, the circuits vary in the number and type of gates, drawing style, and number of loops. Despite these variations, Sketic is able to recognize all gates accurately. Furthermore, as shown in the colored boxes, Sketic is able to recognize all gates with confidence that is close to 99%. We tested our system using a large number of samples collected from students who are taking the digital logic course and in the digital logic lab at the university to make sure that our system does not depend on a certain drawing style or pattern.

**Table 2.** Samples of the outputs using Faster-R-CNN.

Original circuit	After applying Faster-R-CNN
	
	
	
	

### 3.2.3. Artificial neural networks vs. convolutional neural networks

After implementing the detection and the recognition steps using different approaches we concluded that using Faster-R-CNN, which is a CNN-based algorithm, is better for the following reasons:

- Using the CNN algorithms discussed above can detect all circuit elements (gates and inputs/outputs) without the need to have additional image processing steps like drawing bounding boxes and performing the close operation on the circuit elements if an ANN is used.
- CNN-based algorithms can recognize gates NAND, NOR, XNOR, Inverters, and XOR without the need to detect the original gate first (AND, OR, Buffer) and then classify it into its specific type (i.e. classify the gate into AND first and then classify it into AND or NAND) as discussed in the *Detection and recognition using neural networks* section.
- Adding a new gate or input/output to the list of objects to be recognized can be done with negligible effort when CNN-based algorithms are used. However, adding a gate or letter when an ANN algorithm is used requires special handling and we may add an additional stage of classification to resolve issues. Hence, the overhead of the integration is not straightforward.
- The accuracy achieved when CNN-based algorithms are used is higher. Using CNN-based algorithms improved the accuracy to 99% vs. 95% when compared to the results using an ANN.

Table 3 summarizes the differences between the two implemented techniques (ANNs and CNNs) in different design aspects.

**Table 3.** Comparison between NN- and CNN-based classifiers.

Category	NN	CNN
Accuracy	95%	99.5%
Dependency on image processing	Higher	Lower
Use of additional special classifiers and customized heuristics	Higher	Lower
Speed on GPU	Lower speed (requires effort to parallelize the image processing code)	Higher
Overhead of adding new gate and/or letter	Higher	Lower

### 3.2.4. Recognition accuracy and performance analysis

In this subsection, we discuss and compare the recognition accuracy and performance results of famous object detection algorithms against the Faster-R-CNN algorithm used in the present paper. In addition, we compare ANN and faster-R-CNN results against the results of the supervised learning techniques used in recent published work.

**Object detection classifiers:** In addition to the ANN and Faster-R-CNN algorithms discussed before, we tried two widely used object detection algorithms: Yolo [23] and SSD [24]. The two models were trained and tested using the same training and test sets of the ANN and Faster-R-CNN. During the training phase, various configurations were investigated to improve the accuracy and speed of the detection process. The results on the test set show that the accuracy of the Yolo algorithm is 85% and the accuracy of the SSD algorithm is 80%.

The prediction speed (i.e. the time needed to recognize all circuit elements) for the Yolo and SSD algorithms are 1 and 4 s, respectively. On the other hand, the prediction speed for the ANN and Faster-R-CNN algorithms is around 7 and 5 s, respectively. In conclusion, the Faster-R-CNN algorithm is used as the object detection and recognition classifier in Sketic because it achieves the highest accuracy (i.e. 99%) with acceptable 5 s prediction time.

**Supervised learning algorithms:** As mentioned in section 2, recent related work leveraged supervised learning algorithms such as SVM [26] and DT [27] to recognize circuits' components. Such algorithms require a predefined feature vector as an input. The feature vector is a set of metrics that describes the component and distinguishes it from other components. Selecting the right features is critical because it largely affects the accuracy of recognition. In addition, whenever a new circuit component needs to be recognized, the feature vector must be checked to make sure that there are no conflicts with previous components. Otherwise, new features need to be added.

Patara and Joshi [13] used the SVM algorithm to detect digital circuit elements and achieved 83% accuracy. Similarly, Moetesum et al. [15] applied the SVM algorithm to recognize electrical circuit elements and achieved 92% accuracy. On the other hand, Datta et al. [14] used the DT algorithm and achieved 98% accuracy. However, their data set consisted of gate symbols taken from books and simulation tools rather than hand-drawn gates as in our experimental work. Lastly, Dewangan and Dhole [17] used the KNN algorithm and achieved 90% accuracy.

Compared to all the above supervised learning algorithms, leveraging ANN or faster-R-CNN as in Sketic achieves higher accuracy (i.e. 95% for NN and 99% for CNN) on a hand-drawn data set. This is due to the ability of the deep NN to learn concealed structures of objects during the training phase and to extract the required features. As a result, CNNs are widely used for object detection in educational, industrial, and medical fields.

### 3.3. Mapping stage

After all components in the circuit are detected and recognized, we should map connected components: gates, inputs, and outputs. To perform this task the gates and letter rectangles will be extended in the direction of the inputs and the outputs in the case of the gates and in all directions in the case of the letters that represent input and output signals. Then we find the intersection between the extended rectangles and the wire image. If the extended rectangles intersect in the direction of the input and the output of the gate then the wire will be mapped and connected to the gate inside the rectangle. The same procedure will be followed for the inputs and the outputs.

### 3.4. Code generation stage

This stage is responsible for generating the Verilog code that describes the input logic circuit. Before generating the code, letters should be classified into inputs and outputs of the whole circuit. To do that, Sketic takes the outputs of each gate and starts looking for letters connected to the output of the gate. Each letter that is an output of a gate is a circuit output. After all components of the circuit are identified and mapped, Sketic is ready to generate the Verilog code of the circuit.

#### 4. Testing and in-class trials

As mentioned before, such a system will speed up the drafting and the development process in the educational and industrial fields. To enable this in the educational field we applied the system to students taking the digital logic lab course. In the digital logic lab students learn how to use Verilog hardware description language to build digital blocks or modules. In this lab, the students start by drawing the gate level design of the assigned block on paper and then transform this into a Verilog module. Then the students compile the module and download it on FPGAs. To validate our system we asked the students taking the digital logic lab course to draw a random digital circuit that is similar to what they use in the lab. Then we took mobile images for the drawn circuits and we fed them into Sketic to generate the Verilog code.

Table 4 shows a subset of selected samples collected from different students. As shown, the samples cover the basic logic gates and the number of gates in different orders and directions. The second column shows the output of Sketic, where the original circuit is labeled with the result of Sketic detection algorithms. The labeling includes the gate types, wires, inputs, and outputs. For example, if the gate label says “Inverter” then this means that Sketic recognized the gate as an inverter and so on. The yellow labels are for the gates, the red labels are for the input and output variables, and the green labels are for the wires. The wires’ labels are used by Sketic to generate the Verilog code of the entire circuit. The third column in the table shows the corresponding Verilog code for each circuit. The generated Verilog code uses the same names used in the original circuit and defines the unnamed connections as wires.

As shown, Sketic is able to detect all inputs and outputs regardless of their location in the circuit as shown in the first sample. For example, the inputs can be listed on the left side of the circuit and act as inputs to the first level of gates in the circuit or appear in the middle of the circuit as shown in the circuit on the first row. The second sample shows a circuit with buffers and inverters drawn in different directions. As shown, the direction classifier and the gate classifier are able to detect all the gates correctly.

We applied more than a hundred circuits of different sizes and configurations to Sketic and it was able to detect all the circuit elements and generate the Verilog code accurately.

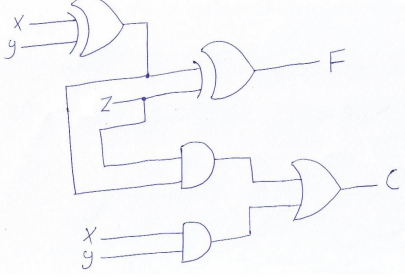
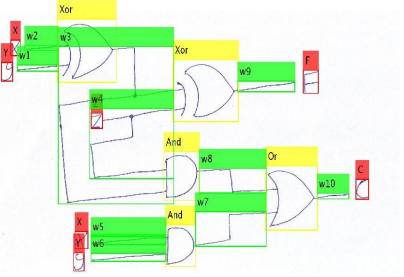
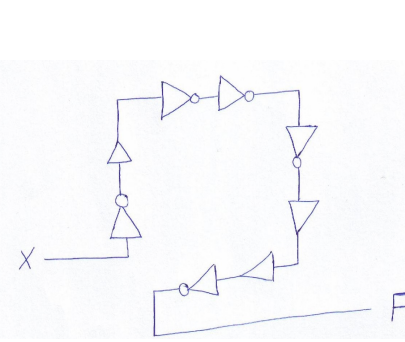
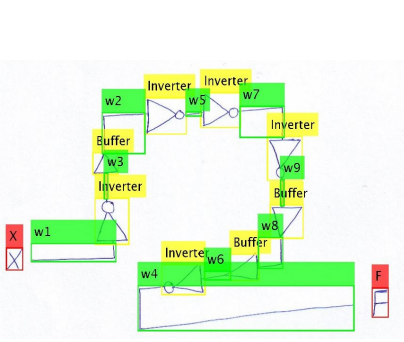
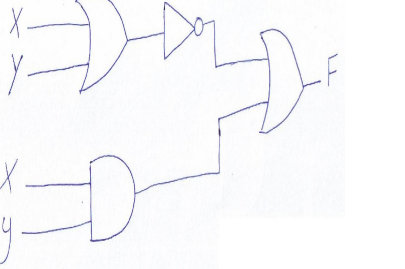
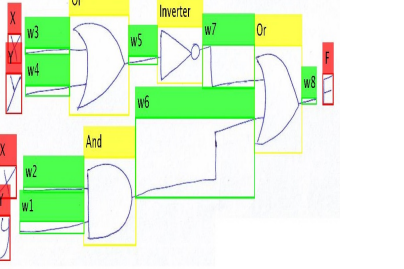
Regarding the time required to generate the results and based on a large number of trials, our system is able to generate the error-free Verilog code in less than 5 s when a laptop with an Nvidia GPU is used. We think that this time is reasonable when compared to the manual approach, which may take tens of seconds or minutes. Moreover, to centralize access the proposed system can be installed on a remote server and the students can upload their images using an online portal to receive the generated Verilog code.

#### 5. Future work

Several steps can be taken to improve and expand the Sketic tool as follows:

- Increase the pool of digital blocks that can be recognized. For example, we can add decoders, encoders, demultiplexers, multiplexers, full adders, and flip-flops. Such blocks are widely used in digital circuit design and they improve Sketic coverage significantly. In the case of conflicts between similar blocks (e.g., demultiplexer and decoder), we can make the tool interactive so that the user is given the option to select the correct logic block to resolve the conflict.
- Apply the same models for recognition of electrical circuits including resistors, capacitors, inductors, transistors, different types of diodes, and voltage and current sources. When integrated with digital circuit recognition, the tool can serve a wider range of users. In the integrated tool, the image of an input

**Table 4.** Samples of Sketic outputs.

Original circuit	Labeled circuit	Generated Verilog code
		<pre> module circuit(Y,X,Z,F,C); input Y,X,Z; output F,C; xor_gate(Y,X,w3); and_gate(w3,Z,w8); and_gate(X,Y,w7); xor_gate(w3,Z,F); or_gate(w7,w8,C); endmodule                     </pre>
		<pre> module circuit(X,F); input X; output F; buffer_gate(w3,w2); inv_gate(X,w3); inv_gate(w2,w5); inv_gate(w6,F); inv_gate(w5,w7); buffer_gate(w8,w6); inv_gate(w7,w9); buffer_gate(w9,w8); endmodule                     </pre>
		<pre> module circuit(Y,X,F); input Y,X; output F; or_gate(X,Y,w5); and_gate(Y,X,w6); inv_gate(w5,w7); or_gate(w6,w7,F); endmodule                     </pre>

circuit is classified as digital or electrical. Afterwards, all elements in the circuit are recognized using the developed classifier.

- Add an option to generate an image of the recognized circuit using clean predefined gates' symbols. The output image can be used by instructors in preparing exam questions and PowerPoint slides quickly without the need to manually draw the circuit using painting tools.
- Add an option to generate the transistor level netlist of the recognized digital circuit. This will make the transition time from a sketch to a full circuit minimal.
- Develop a mobile application and a web portal for the tool, such that the circuit image can be easily uploaded from a mobile phone and the output files are sent to the user's preferred email.

## 6. Conclusions

In this paper, we were able to design an application capable of detecting logic circuits with 99% accuracy. In Sketic, we relied on the emerging machine learning algorithms and image processing toolboxes to build the

application. In the proposed solution we divided the work into four steps, where each step takes the output from the previous step and gives its output to the next step. In the recognition part we relied on the emerging machine learning algorithms to make sure that our system can work accurately with different drawing styles. In the recognition stage we applied two different algorithms (ANN- and CNN-based algorithms) to build the system. These neural networks (ANNs and CNNs) are surrounded by the preprocessing, mapping, and code generation steps to implement the target functionality of generating an accurate Verilog code that represents the circuit between hands. While both algorithms have more than 90% accuracy we noted that the Faster-R-CNN algorithm is better than the ANN-based structures in terms of recognition accuracy.

At the end our system is able to recognize full digital circuits with almost perfect accuracy. We tested the system on samples collected from students taking the digital logic lab course and the system shows 99% accuracy in detecting and recognizing the drawn circuits.

While in this paper we focused on digital logic circuits, the proposed techniques can be extended to any type of sketched circuits or structures with predefined basic blocks. RLC circuits and transistor level designs like amplifiers and filters are some examples of circuits that can be detected and recognized by our proposed approach with minimum effort.

## References

- [1] Intel Corporation. Simulation quick-start for ModelSim\* - Intel FPGA edition. White paper, 2017.
- [2] Palnitkar S. Verilog HDL: a guide to digital design and synthesis. Englewood Cliffs, NJ, USA: Prentice-Hall, 1996.
- [3] Liwicki M, Knipping L. Recognizing and simulating sketched logic circuits. In: Proceedings of the Ninth International Conference on Knowledge-based Intelligent Information and Engineering Systems; Melbourne, Australia; 2005. pp. 588-594.
- [4] Burch C. Logisim: a graphical system for logic circuit design and simulation. *Journal on Educational Resources in Computing* 2002; 2 (1): 5-16. doi: 10.1145/545197.545199
- [5] Alvarado C, Davis R. Sketch READ: a multi-domain sketch recognition engine. In: Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology; Santa Fe, NM, USA; 2004. pp. 23-32.
- [6] Alvarado C, Kearney A, Keizur A, Loncaric C, Parker M et al. LogiSketch: a free-sketch digital circuit design and simulation system. In: Proceedings of the Workshop on the Impact of Pen and Touch Technologies in Education; Los Angeles, CA, USA; 2013. pp. 83-90.
- [7] Zamora S, Eyjólfssdóttir E. CircuitBoard: sketch-based circuit design and analysis. In: Proceedings of International Conference on Intelligent User Interfaces (IUI) Workshop on Sketch Recognition; Sanibel Island, FL, USA; 2009.
- [8] Alzubi J, Nayyar A, Kumar A. Machine learning from theory to algorithms: an overview. *Journal of Physics: Conference Series* 2018; 1142: 012012. doi: 10.1088/1742-6596/1142/1/012012
- [9] Saifan R, Dweik W, Abdel-Majeed M. A machine learning based deaf assistance digital system. *Computer Application in Engineering Education* 2018; 26 (4): 1008-1019. doi: 10.1002/cae.21952
- [10] Kia M, Alzubi J, Gheisari M, Zhang X, Rahimi M et al. A novel method for recognition of Persian alphabet by using fuzzy neural network. *IEEE Access* 2018; 6: 1-20. doi: 10.1109/ACCESS.2018.2881050
- [11] Boutaba R, Salahuddin MA, Limam N, Ayoubi S, Shariar N et al. A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Journal of Internet Services and Applications* 2018; 9 (16): 1-100. doi: 10.1186/s13174-018-0087-2
- [12] Ma S, Sun Y, Lyu P, Polsley S, Hammond T. DCSR: a digital circuit sketch recognition system for education. In: Hammond T, Adler A, Prasad M (editors). *Frontiers in Pen and Touch*. New York, NY, USA: Springer, 2017. pp. 137-146.

- [13] Patare M, Joshi M. Hand-drawn digital logic circuit component recognition using SVM. *International Journal of Computer Applications* 2016; 143 (3): 24-28. doi: 10.5120/ijca2016910058
- [14] Datta R, De P, Mandal S, Chanda B. Detection and identification of logic gates from document images using mathematical morphology. In: *Fifth National Conference on Computer Vision, Pattern Recognition, Image Processing and Graphics*; Patna, India; 2015. pp 1-4.
- [15] Moetesum M, Younus S, Warsi M, Siddiqi I. Segmentation and recognition of electronic components in hand-drawn circuit diagrams. *EAI Endorsed Transactions on Scalable Information Systems* 2018; 5 (16): 1-6. doi: 10.4108/eai.13-4-2018.154478
- [16] Lakshman R, Dinesh R, Prabhanjan S. Handwritten electric circuit diagram recognition: an approach based on finite state machine. *International Journal of Machine Learning and Computing* 2019; 9 (3): 374-380. doi: 10.18178/ijmlc.2019.9.3.813
- [17] Dewangan A, Dhole A. KNN based hand drawn electrical circuit recognition. *International Journal for Research in Applied Science and Engineering Technology* 2018; 6 (6): 1111-1115. doi: 10.22214/ijraset.2018.6164
- [18] Grother P, NIST Special Database 19. *NIST Handprinted Forms and Characters Database*; Gaithersburg, MD, USA; 1995.
- [19] MATLAB and Statistics Toolbox Release 2012b, The MathWorks, Inc.; Natick, MA, USA; 2012.
- [20] Peng C, Lee K, Ingersoll G. An introduction to logistic regression analysis and reporting. *The Journal of Educational Research* 2002; 96 (1): 3-14. doi: 10.1080/00220670209598786
- [21] Yao X. Evolving artificial neural networks. *Proceedings of the IEEE* 1999; 87 (9): 1423-1447. doi: 10.1109/5.784219
- [22] Qian H, Xu J, Zhou J. Object detection using deep convolutional neural networks. *Chinese Automation Congress (CAC)*; Xi'an, China; 2018. pp. 1151-1156.
- [23] Redmon J, Divvala S, Girshick R, Farhadi A. You only look once: unified, real-time object detection. In: *The IEEE Conference on Computer Vision and Pattern Recognition*; Long Beach, CA, USA; 2016. pp. 779-788.
- [24] Liu W, Anguelov D, Erhan D, Szegedy C, Reed S et al. SSD: single shot multibox detector. In: *European Conference on Computer Vision*; Amsterdam, Netherlands; 2016. pp. 21-37.
- [25] Ren S, He K, Girshick R, Sun J. Faster R-CNN: towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 2017; 39: 1137-1149. doi: 10.1109/TPAMI.2016.2577031
- [26] Burges C. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery* 1998; 2 (2): 121-167. doi: 10.1023/A:1009715923555
- [27] Gehrke J, Ramakrishnan R, Ganti V. Rainforest - a framework for fast decision tree construction of large datasets. In: *Proceedings of 24th International Conference on Very Large Data Bases*; New York, NY, USA; 1998. pp. 416-427.