From Component Detection to Simulation of Analog Circuits Using CNNs

Matthildur Margret Arnadottir
mma29@stanford.edu
SUNet ID: 006655070

Ottar Yngvason
ottar@stanford.edu
SUNet ID: 06645575

Abstract

Processes in industries today are becoming increasingly more automated. Despite this, the process of obtaining simulations of a hand-drawn electric circuit schematic is usually done by hand. In this project, an algorithm was designed to detect circuit from a hand-drawn photo and generate a simulation of the circuit. The algorithm uses a convolutional neural network for classification of components and sliding windows for their detection.

1 Introduction

The aim of this project is to accurately detect each component of a hand-drawn electronic circuit. For a given image, the goal is to identify each component and return the image with appropriate bounding boxes along with the class of each component. Furthermore, for a particular type of circuit, this method makes it possible to produce code that can be directly exported to *HSpice* (a popular circuit simulation software) for simulation. We will only be using discrete analog circuits in this project. We chose this project because it aligns with our interests in electrical engineering, and it could have practical applications since an immediate simulation of a hand drawn diagram is tremendously useful.

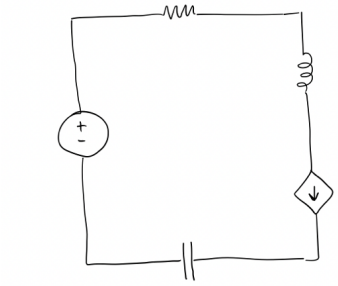

Figure 1: Circuit Schematic (input).

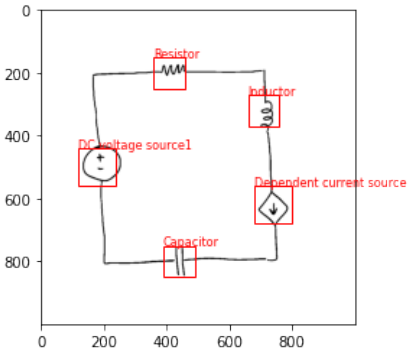

Figure 2: Circuit with Correctly Detected Components (output).

2 Related work

Not many papers have been written on this particular topic. However, of the few published papers that relate to this topic, one described a project quite similar to this one. Still, it did not use deep learning methods, instead an SVM classifier was used [1]. This paper was ambitious in its approach

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

but did not emphasize the practical applications of the algorithm. Other implementations have used deep learning methods but only for component classification, but not detection [4], [9]. One such method used feature extraction by using image moments to reduce the number of features [6]. This approach was relevant because the algorithm used a dense neural network instead of a convolutional neural network. Finally, one paper achieved high accuracy using traditional learning algorithms and also included digital circuit components such as logic gates [8]. However, this algorithm also did not include component detection.

3 Dataset

For training a neural network to classify components, we used a dataset from Kaggle [3], which contains 14 classes of components, with around 200 pictures of each class. This is not a very large dataset, and requires some preprocessing. We began by inverting the images since they are drawn with a white pen on a dark background, but most hand-drawn circuits are drawn in dark on a white background. The images were also converted from RGB to grayscale to reduce computation cost. The images were then reduced from the size 120x120 down to 64x64, also to reduce computation. The original images only had pixel values 0 or 255. However the resize method used set some pixels to intermediate values. Dark intermediate pixels of the resized images were set to 0 while the rest were set to 255. To increase the size of the dataset, we augmented the data by spinning each image by 90, 180 and 270 degrees. This not only quadrupled the size of the dataset, but is also very logical since circuit components are drawn at a variety of different angles in practice. We also added about 800 blank images so the classifier could have a "no component" output. Finally, since the ultimate goal was to detect these classes in a circuit, we added another 800 images of "wires", which we scraped out of numerous circuit diagrams. After augmentation, and the addition of the blank and wire images, the dataset contained 12,656 labeled images. We split the dataset 70/15/15 between training, validation and test sets. To test the component detection algorithm, 20 pictures of circuits were drawn. For simplicity, the circuits were drawn on a tablet.

4 Component classification

4.1 Method

For component classification we used a neural network with four convolutional layers, four pooling layers and three dense layers with the output layer being a softmax layer. All of the filters were 3-by-3. We designed our baseline model as a variation of a model designed to recognize digits in the MNIST dataset, [5] by Sambit Mahapatra. We decided to base our model on this particular CNN because it achieved over 99% accuracy and works therefore well on classifying images of handwritten objects. The images in our dataset are bigger than the ones in the MNIST dataset, so we added two more convolutional layers. To avoid overfitting, we added more Dropout layers as well. The network architecture is shown in figure 3.

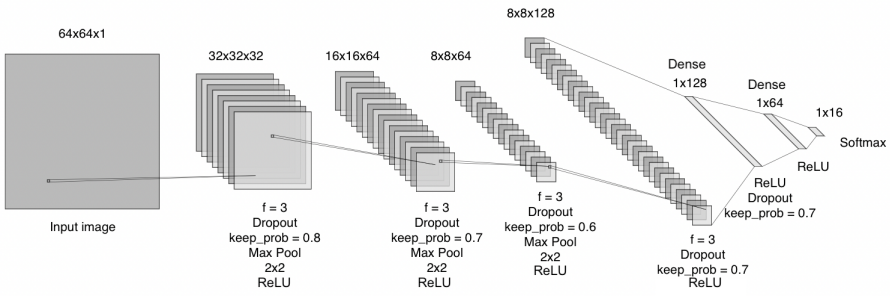

Figure 3: Network architecture.

We trained the network with ADAM optimizer and a batch size of 32. We used the standard Keras categorical crossentropy loss function and an accuracy metric in training.

4.2 Model variations

Five variations, with different hyperparameters, of the model shown above were tried. Each was trained for 50 epochs. Table 1 shows the different models that were evaluated. The table also shows that adding dropout did in fact reduce overfitting. Since it has the highest accuracy, the original model in figure 3 will be used in the proceding discussion.

Model	Test Accuracy
No change	0.9543
Add two Conv2D layers (f = 5, no pooling, dropout = 0.3, padding = same)	0.9132
Remove the smaller dense layer	0.9353
Add batch normalization	0.6128
Remove dropout	0.9152

Table 1: Different variations of the model.

4.3 Results

Accuracy was the metric used for evaluation of the classification algorithm. The neural network in figure 3 achieved over 98% training accuracy and 94% validation accuracy. A confusion matrix for the test set was constructed for the test set and it is shown in figure 4. The code for the confusion matrix formatting was borrowed from the web [10]. As seen from the confusion matrix, the most

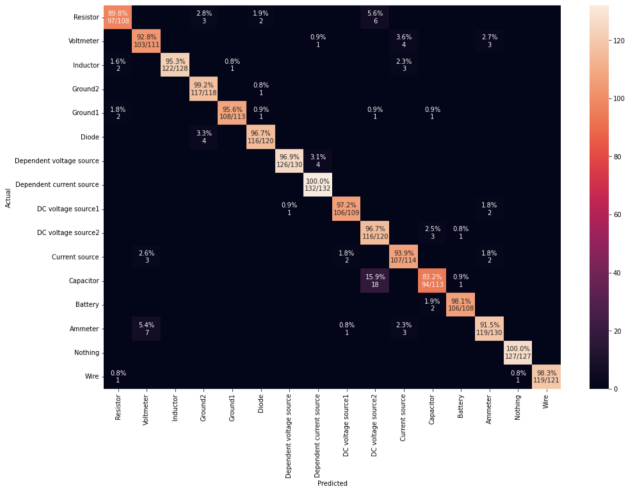

Figure 4: Confusion matrix for the classifier.

common mistake the network makes is misclassifying DC voltage sources as capacitors. This is also a problem for humans since it is not always clear if a component is a capacitor or a DC voltage source since the only difference is the size of one of the lines.

5 Component detection

5.1 Method

After the model was trained to classify components, it could then be used to detect components in a circuit diagram. Without a dataset of labeled bounding boxes, the best approach was with a sliding-windows method. This algorithm scans an input image, and predicts the contents of each window. If the prediction is not "Nothing" or "Wire", and the highest probability is over a certain threshold, then a bounding box is placed on the window. A threshold in the range $0.6 - 0.7$ was found to provide the best results. We drew inspiration from a sliding windows implementation we found on the web [7].

The size of the window matters greatly, as well as the size of each step. The window size must be larger than the components, but not so large that two components could be confused together. A small step size is computationally expensive, but if it is too large then some components could be missed. Therefore, the algorithm standardizes each input image to a size 1000x1000 and tests out three different sets of window and step sizes that have proven to work well in practice. The window sizes we settled on were 100, 120, and 140 and the corresponding window sizes were 30, 40 and 50.

To filter out overlapping boxes, the algorithm then performs non-max suppression. The boxes are sorted by their scores, which we define as the highest classifying probability. [2] It is then possible to calculate the intersection-over-union ratio of the box with the highest probability and each of the other boxes, removing those that overlap. Iterating over all boxes and removing those with lower probabilities should leave us with only the correct boxes. For this use case, we can impose this strict threshold of 0, meaning that no overlapping is allowed.

5.2 Results

To evaluate the accuracy of this method, a dataset of 22 images of hand-drawn circuits was used, with each circuit containing 2 to 5 components. Without a labeled test set, the most descriptive method of evaluation is defined as follows.

$$\text{Testing accuracy} = \frac{\#\text{ of correct boxes}}{\text{Total }\#\text{ of boxes predicted}}$$

Running the code on each image in the test set, the algorithm achieves 79.4% accuracy as defined above. Some classes are easier to detect than others, which correlates with the results of the classifier. While resistors are always detected with 100% accuracy and classified correctly, DC voltage sources and capacitors are frequently mixed up, and it is also hard for the model to differentiate between dependent and independent sources. More specifically, current source detection only reaches an accuracy of 50%, and dependent current source detection accuracy was 25%. To increase detection accuracy, it would be practical to add more of these hard examples to the training set.

6 Simulation of parallel circuits

One of the biggest practical uses of this algorithm would be to directly produce a simulation of a circuit from a hand-drawn image. In today's schools and even industry, a substantial amount of time is spent typing in the parameters of a hand-drawn schematic of a circuit into simulation software. Therefore, automating this process would be extremely useful.

6.1 Method

Electrical engineers commonly use a simulation software called *HSPICE* for analog circuits. *HSPICE* can perform a wide variety of simulations for electric circuits. The software takes in a so-called netlist, which documents how components are connected and the component values.

For this algorithm to work on general circuits, node detection would need to be implemented for the diagram of the circuit. This proved to be a daunting task. Therefore, simply to show how this method could be used, we decided to restrict the circuits simulated to parallel circuits. Parallel circuits are circuits where there are only two nodes and all the components are connected between those nodes.

Code was written, to generate a netlist for a circuit, whose components had been detected with the component detection algorithm previously described. The only components allowed were independent current and voltage sources, resistors, capacitors and inductors. Originally the plan was to detect the values of the circuit components in the diagram as they are commonly written close to the component. However, this also proved to be a daunting task. Therefore, it was decided to make the user submit the values of the components from left to right. A netlist can be run in *HSpice* to simulate the circuit.

6.2 Results

We decided to perform an AC sweep of the circuit from 1 Hz to 10 GHz to make a Bode plot of the circuit's transfer function. The Bode plot was made with software called *CosmosScope* which

can visualize an *HSpice* simulation. Below is an example of the amplitude and phase response for one parallel circuit. The resistor value was set to 10k and the other components were set to 1 for simplicity.

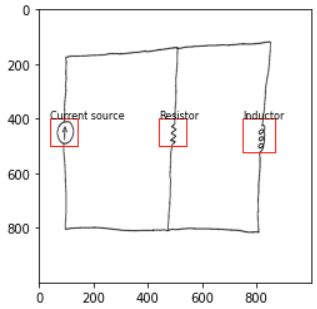

Figure 5: Circuit with detected components.

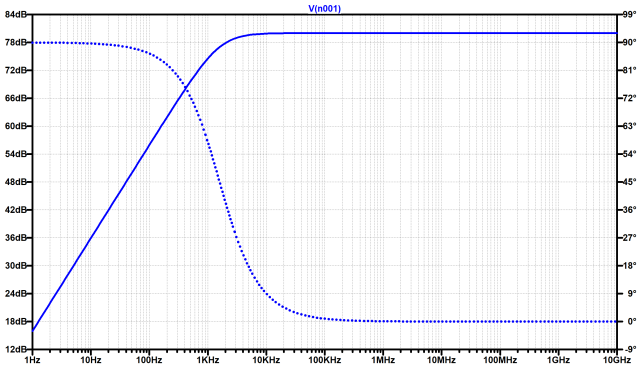

Figure 6: Frequency and amplitude response for the circuit in figure 5.

7 Discussion

We faced many challenges working on this project. During initial testing on our sliding windows algorithm, we noticed that it was frequently misclassifying wires as DC voltage sources, capacitors and other components. Therefore, we decided to directly add the images of wires misclassified as other components to our dataset and created a new category for wires. After retraining the model with these images, this problem subsided.

With more time, certain advancements to this project are possible. The next step would be to add value detection of hand-drawn numbers for each component. Furthermore, by adding nodal detection, the *HSpice* code generation method could be generalized to a far broader class of analog circuits.

Both the performance and the running time of this algorithm could likely be drastically improved by using a fully convolutional method like YOLO. Using YOLO was our original intention. However, this algorithm would require a dataset with labeled bounding boxes, which was not to be found on the web.

8 Conclusion

The aim of this project was to detect components in a circuit, and the method proposed in this report is capable of doing exactly that with nearly 80% accuracy. With more hard examples added to the dataset and further hyperparameter tuning and the classifier could possibly improve substantially.

This algorithm might also aid in the automation of circuit design and analysis. Accurate and labeled bounding boxes provide the coordinates of each component in an image. The relative connections could be calculated with more advanced code for nodal detection. This would allow for a broader class of circuits to be simulated using the method developed in this paper.

9 Team member contributions

We worked on the code in close collaboration. Matthildur took the lead on data preprocessing and implementing non-max suppression. Ottar took the lead on error analysis, implementation of sliding windows and simulation. Both team members have contributed to designing the network architecture, as well as documenting the process in the report, video, and code.

References

[1] Mahmut Aksakalli. *Hand-drawn Circuit Recognizer*. URL: https://github.com/mahmut-aksakalli/circuit_recognizer?fbclid=IwAR28Pv843zmWt4B0KjKN1TjXLTTjY981EuQBpI3C6NPocbF47Ll0rHqgFVUhttps://towardsdatascience.com/a-simple-2d-cnn-for-mnist-digit-recognition-a998dbc1e79a. (accessed: 11.11.2022).

[2] CS230 faculty. *Car detection with YOLO*. URL: https://www.coursera.org/learn/convolutional-neural-networks/programming/3VCFG/car-detection-with-yolo. (accessed: 12.8.2022).

[3] Mahmoud Gody. *Hand-drawn Electric circuit Schematic Components*. URL: https://www.kaggle.com/datasets/moodrammer/handdrawn-circuit-schematic-components. (accessed: 10.14.2022).

[4] Mihriban Günay, Murat Köseoğlu, and Özal Yıldırım. *Classification of Hand-Drawn Basic Circuit Components Using Convolutional Neural Networks*. URL: https://ieeexplore.ieee.org/document/9152866/authors#authors. (accessed: 11.30.2022).

[5] Sambit Mahapatra. *A simple 2D CNN for MNIST digit recognition*. URL: https://towardsdatascience.com/a-simple-2d-cnn-for-mnist-digit-recognition-a998dbc1e79a. (accessed: 10.14.2022).

[6] Mahdi Rabbania et al. *Hand Drawn Optical Circuit Recognition*. URL: https://www.researchgate.net/publication/302980429_Hand_Drawn_Optical_Circuit_Recognition. (accessed: 12.05.2022).

[7] Adrian Rosebrock. *Sliding Windows for Object Detection with Python and OpenCV*. URL: https://pyimagesearch.com/2015/03/23/sliding-windows-for-object-detection-with-python-and-opencv/. (accessed: 11.07.2022).

[8] Soham Roy et al. *Offline hand-drawn circuit component recognition using texture and shape-based features*. URL: https://link.springer.com/article/10.1007/s11042-020-09570-6. (accessed: 11.23.2022).

[9] Haiyan Wang1, Tianhong Pan2, and Mian Khuram Ahsan. *Hand-drawn electronic component recognition using deep learning algorithm*. URL: https://www.inderscienceonline.com/doi/pdf/10.1504/IJCAT.2020.103905. (accessed: 11.06.2022).

[10] Runqi Yang. *Generate matrix plot for confusion matrix with pretty annotations.* URL: https://gist.github.com/hitvoice/36cf44689065ca9b927431546381a3f7. (accessed: 12.07.2022).