

“CircuitMagic”

Written By: Ed Pataky, Autodesk Inc.

Automatic Capture of Hand-drawn Electronic Symbols and Component Selection in an Electronic EDA CAD System using Machine Learning Techniques

Ed Pataky, San Francisco, California

Application Number: ----

Inventors: Edward Sandor Pataky

Technical field

This disclosure is related to computer simulation tools, electronics CAD, Software, Software Systems, Machine Learning Techniques, Algorithms.

Background

Consider a modern client-server/cloud eda cad design system as shown in *Fig. 1*. Multiple clients use a front-end client application (the cad system interface software program) on their devices (mobile, desktop, laptop, etc) to connect to a cloud back-end system which performs various tasks. In modern systems, like Autodesk 123d Circuits or iSchematics' Spicy Schematics, in fact many thousands of users connect to the cloud system, and the front-end interface is a software program that runs in the client's internet browser. The backend cloud system handles various operations, including managing user accounts and design files, and real-time storage and retrieval of user data.

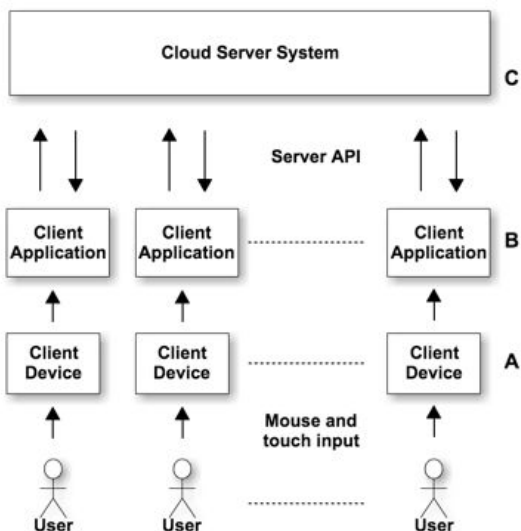


Figure 1: Client-Server/Cloud EDA CAD System

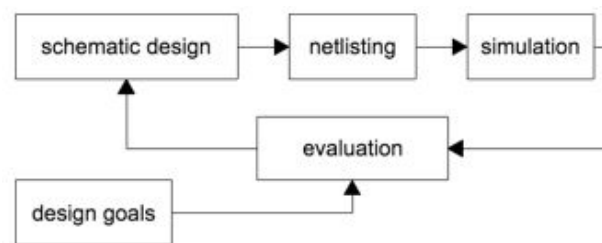


Figure 2: Typical Circuit Design Flow using EDA CAD Tools

“CircuitMagic”

Written By: Ed Pataky, Autodesk Inc.

In a typical EDA circuit CAD system, the user design flow (shown in *Fig. 2*) would be to first enter the circuit schematic in a schematic capture application, then (optionally) create a netlist and finally to simulate the circuit. Once the circuit is simulated, in what we will call the evaluation phase, the user evaluates the simulation results, given their design goal for the circuit, and decides whether or not to iterate the design by making changes in the schematic and repeating the steps. Once the simulation results are favorable, the user will decide the design is complete and may (optionally) move on to a physical printed circuit board layout tool, or other simulation or design tool.

Note that especially because the design involves iterating the design flow loop from *Fig. 2*, a good deal of time is spent in the schematic entry step. The schematic entry step must as simple as possible for the user, to minimize time spent.

In a modern web-based circuit CAD system, shown in *Fig. 3*, the cloud server system consists of multiple servers, optionally organized by function, and users connect to the cloud via a client application which contains the user interface (UI) where they enter the schematic.

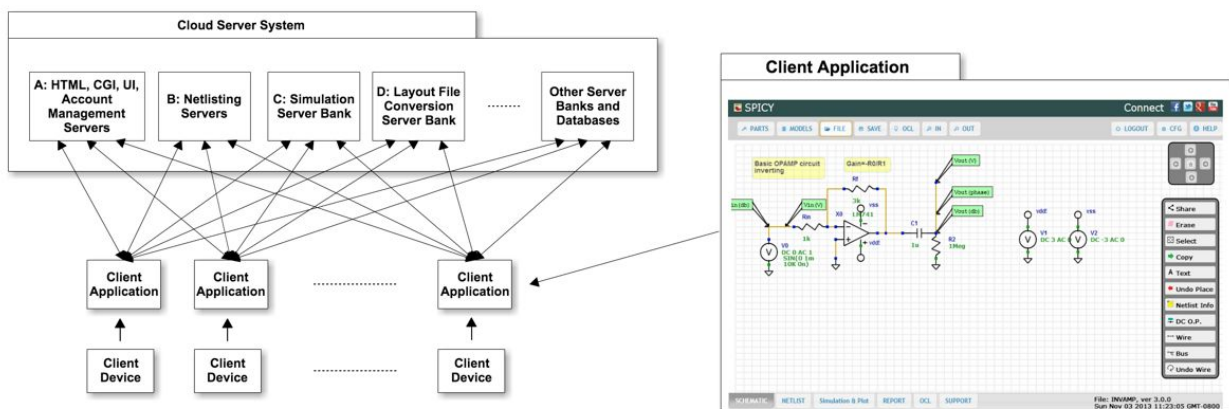


Figure 3: Client-Server/Cloud EDA CAD System - Architecture

The details of the UI may change, but in general, there is some hierarchy of menus the user has to navigate to find components to place as they build their schematic. For example in the Spicy Schematics system shown in *Fig. 3*, The user would click on menu labeled “Parts”, then a submenu labeled “Spice”, then they would choose parts for a spice type simulation from additional submenus such as “Passives”, and “Actives”. Some application UI designs contain menus that are designed to be “flat” (for example all components are shown in a list at one level), the user does not need to select from too many submenus, but still they would need to scroll through the list of options to find the part they are interested in each time they place a component. In the case of very commonly used parts, like resistors and capacitors, this can become very repetitive for the user and forces them to spend too much time on creating the schematic, than on perfecting their design. Better-designed applications can use assistive features such as sorting components by last used, and/or frequently used, and the UI can support the user typing in a description to more easily find a part, but in the end, compared to

“CircuitMagic”

Written By: Ed Pataky, Autodesk Inc.

hand-drawing a schematic, the selection and specification of components in virtually any eda cad tool is still inherently a task with many steps.

For users that access the client application via mobile tablet computers, where much of the input is done by touch (either by finger or by stylus), it can be doubly frustrating to the user when they have to repeat operations, for example in the case of tapping on the wrong item or in the wrong location. Because touch-based technology can typically be inherently less precise than using a mouse in a traditional computer environment, touch-based applications have to be even more diligent in providing an intuitive user interface than those on traditional computer systems.

Arguably today, with application development and distribution democratized by platforms such as Apple’s iTunes, which makes available millions of applications from many thousands of developers, in fact if a user feels an application is not as intuitive as they expect, they simply delete the application and download another. In this way, an application can live or die in an instant based on the first time a user uses the application, as they judge the user interface to be either easy or hard to use. Only those that have made a concerted effort to create a smooth user experience will survive and be successful. In this way, user interface design for touch-based systems is not only important, it is a critical factor in the success of a product.

To make schematic entry simpler for the user, what would be ideal is to allow the user to quickly sketch the component they want, and have the system recognize it and place the matched part, or at a minimum, automatically show a list of components that are similar for the user to quickly choose from.

Consider the mockup user-interface shown in Fig. 4, using the Spicy Schematic system as a base, and consider we are using the application from within a touch-input-enabled mobile device such as an Apple iPad. The UI could include a small drawing area (titled “Draw Component” in Fig. 4) where the user could draw the symbol they want to place. This small hand-drawing area could be open and visible at all times, or, as inferred in the figure, could be opened by the user tapping a button that is always prominently visible on the UI. The user opens the drawing area, sketches the part they want to place, and the system would send the captured image data to the cloud for processing. Once processed, the server would send back to the UI a value that indicates to the UI which UI component (or list of components) best match the user’s drawing. If there is a match, the application UI can automatically place the component, optionally opening up properties or other menus to allow the user to continue to further specify the component specification within the schematic. In this way, schematic entry would be greatly simplified, and the need for menus or manual typing of descriptions to find components would be eliminated. This would save the user a great deal of time as they build their schematic, and they will be able to reach their design goal faster.

“CircuitMagic”

Written By: Ed Pataky, Autodesk Inc.

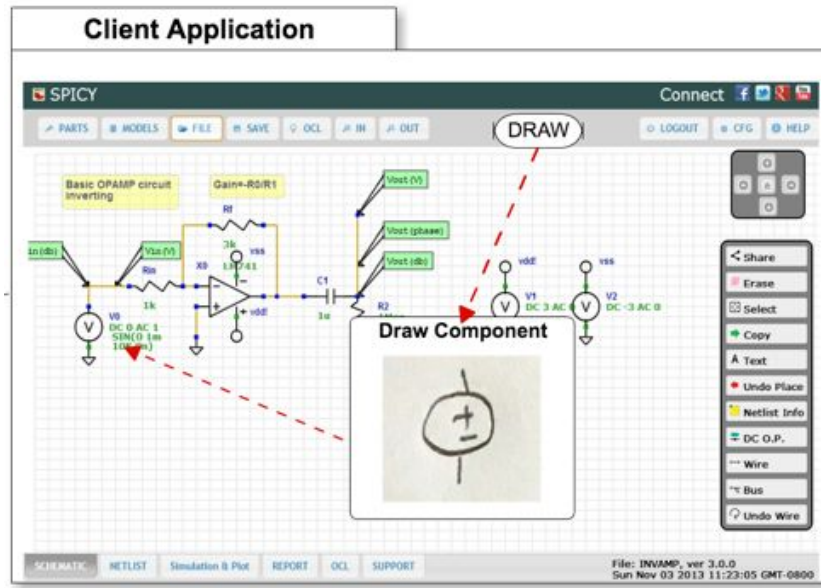


Figure 4: UI for Circuit EDA Cad System with Hand Drawing Recognition

To make this a reality, The back-end cloud system can run an artificial neural network (ANN) machine learning (ML) algorithm that has been trained on the available components, and is able to recognize hand-drawn components. This is very similar to the widely known ANN hand-drawn decimal digits recognition ML problem, in which a trained neural network is used to decipher hand-drawn decimal digits.

To understand how this can be made a reality, we look at the problem in three parts:

1. The client-server eda cad **system architecture** required
2. The front-end **user interface** requirements, and
3. The machine learning **algorithms** required on the back-end

“CircuitMagic”

Written By: Ed Pataky, Autodesk Inc.

System Architecture

Consider again the system shown in Fig. 1, and consider that a user runs the client application B on their mobile device A. The application communicates in both directions with the cloud server system C, assuming the user is connected to the internet. The mobile device A is a computing device that can run the schematics entry program, and has internet connection capability so that it is able to communicate with other computers, such as a cloud server system. The cloud server system C is a computer, or set of computers, that is configured to communicate with the client application B, and may optionally be able to communicate with other computers, databases, and filesystems.

In Fig. 5 we see this same architecture with more detail on how data flows from the client to the server for hand-drawn component recognition. Here we see the client application A sends the drawn symbol data B to the cloud server D. The cloud server D communicates with the client application A via the server back-end software F. The back-end software F has access to a trained algorithm E as well as (optionally) a database G. We show here the back-end software as separate from the algorithm E, because we are showing the client-server communication management software as a separate module or program, as would typically be the case, although it is not strictly required for the two programs to be separate.

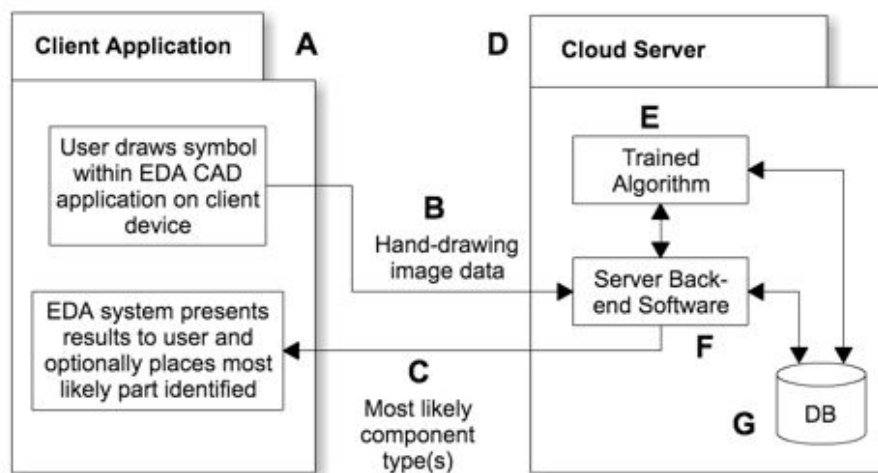


Figure 5: System Architecture and Logical Flow

Server back-end software as an endpoint for client-server communication typically has multiple functions (as in Fig. 3), including verification of the user account, and data storage management.

“CircuitMagic”

Written By: Ed Pataky, Autodesk Inc.

As an example, the Spicy Schematics client application created by iSchematics LLC is a web-based cad application that can run on an Apple iPad device. The iPad device has the ability to connect to the internet, and allows the client application to communicate with special servers that are configured to: (a) login users and give them access to their files on the server, (b) take schematic data and produce so-called spice-format circuit netlists, (c) create netlists and run circuit simulation programs on the cloud server computers, (d) return circuit simulation results and tell the client application to plot and display those results, and more. In the Spicy system, the back-end software is written in Python, the client application in HTML, CSS and Javascript, and there are sqlite and mysql databases. Note that, as in Spicy, the communication channels shown in Fig. 6 (B and C) could be in reality be a single communication channel, that works by providing an application programming interface (API) for the client application to send and receive data from the cloud servers.

User Interface

The user interface requirements for hand-drawn electronic symbol recognition are as shown in Fig. 4 and Fig. 6. From within an electronic EDA CAD schematics entry application, the user needs to have the ability to hand-draw a schematic symbol. This can be presented to the user in many forms, and could be accessed in many different ways. The drawing window can be permanently part of the interface, visible at all times, for example, or it can be opened by the user inputting some action to the interface such as a button press or menu selection. In the preferred embodiment, to maximize the space on screen for the user to view the schematic, the user would click a button to open the drawing window, which would stay visible and active until the user closes it.

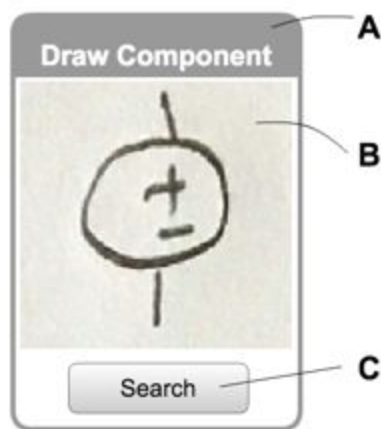


Figure 6: Hand-Drawing UI Window

Fig. 6. Shows the basic components of the UI window for the user to draw components. The overall window (A) is titled “Draw Component” and contains an area (B) for user input, where the user would draw the symbol they wish to place. The user presses/taps button (C) when they are done drawing, to tell the system to process the input. Note that the button is optional and algorithms could be developed to automatically detect when the user is done drawing instead of requiring the user to press a button when done. In addition, there could be other optional buttons made available to allow the user to, for example, clear the drawing area, choose colors, or perform other operations.

“CircuitMagic”

Written By: Ed Pataky, Autodesk Inc.

Back-End Algorithms

Artificial Neural Network Model

From Fig. 5 we saw that the cloud server(s) E use a trained algorithm F to process the incoming hand-drawn image data B. This refers to an Artificial Neural Network (ANN) software program that has been trained to recognize hand-drawn schematic symbols. By Artificial Neural Network, here it is meant as the most general description that includes single and multi-layer networks, Convolutional Neural Networks (CNN), and other topologies.

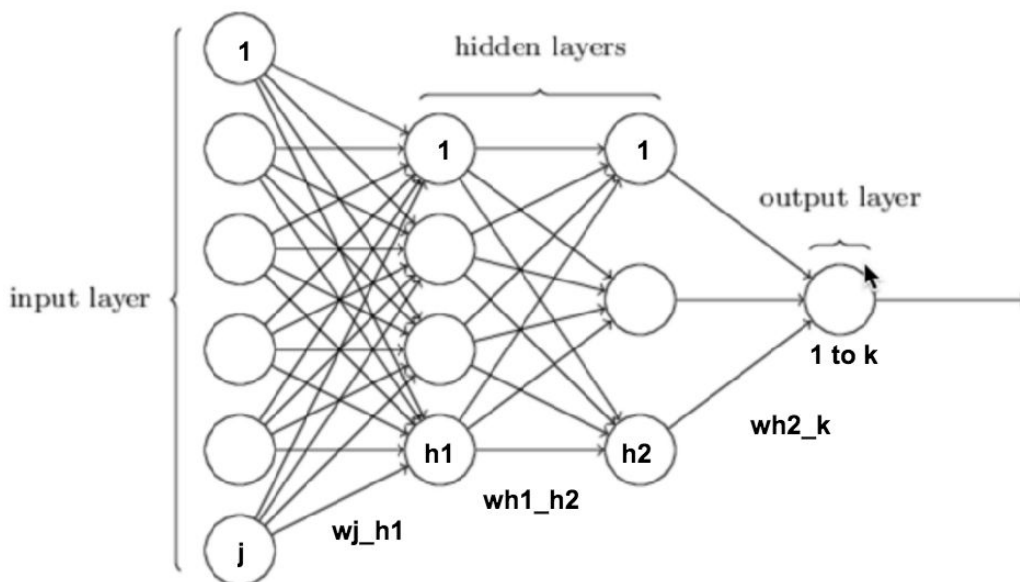


Figure 7: Multi-Layer Artificial Neural Network Model

Fig. 7 shows a general multi-layer artificial neural network model with 6 input nodes and 1 output node, and 2 hidden layers with 4 and 3 nodes, respectively from left to right. Each node performs a specific mathematical function on its inputs (called its activation function for hidden layers), and there are weights associated with the connections between layers. Weights w_{j_h1} represents the weights from input node j to hidden layer 1 node $h1$, where j is number from 1 to the number of input nodes, and $h1$ is a number from 1 to the number of nodes in the first hidden layer. Weights w_{h1_h2} represents the weights from hidden layer 1 node $h1$ to hidden layer 2 node $h2$, where $h1$ is number from 1 to the number of nodes in the first hidden layer, and similarly for $h2$. Weights w_{h2_k} represents the weights from hidden layer 2 node $h2$ to output node k , where k is a number from 1 to the number of output layer nodes. It is noted that not shown in the figure, but assumed and used in the preferred embodiment here, is an additional output layer called a “softmax” layer, in which there is one node for each “output” node, which calculates a ratio of each output nodes’ output to all other output nodes’ outputs, and indicates which output is the most likely class for the input data.

“CircuitMagic”

Written By: Ed Pataky, Autodesk Inc.

Note that this is a general representation of a neural network. Specific model implementations specify how many connections there are, which nodes are connected to each other, how many layers are used, and what activation functions are used.

In this invention, we consider image data input with width and height sizes $[w,h]$, and total number of pixels $p = w \cdot h$. We consider an ANN model consisting of 1 input layer with p nodes, 1 hidden layer with h nodes (more shortly), and an output layer with N nodes, where N is the number of types of symbols that we want to recognize. As a machine learning problem, this is setup and appropriately termed as a classification problem, where we want to classify the incoming hand-drawn data as being an instance of one of the known ‘classes’ of inputs. In the case of an electronic eda cad system where the ANN is used to classify hand-drawn symbols, the classes are the known symbols that the system is to recognize.

It is noted that for the system to have a high accuracy, the network should be able to recognize hand-drawn symbols, regardless of rotation, translation, and scale, since not everyone will draw symbols perfectly. These invariances are addressed in the way the network is trained, and more specifically, in terms of what inputs are given to the network while being trained (called the training data). More on this later.

In this context, that of a supervised classification machine learning problem using a multi-layer neural network model, algorithm training is the process where classified data (samples of data where the expected class is known) is input to the neural network, and the connection weights between node layers are found such that the classification error is minimum, given the network topology and node functionalities chosen. If a network is trained properly and is successful, given the weights from an already trained network, the network can be ‘activated’ on a sample data point (in this context a sample of data is one full image), and the output error can be expected to be roughly what was seen during training.

The way the back-end classification processing works is that first, image data is received on the cloud server, having been sent from a schematic entry client application running on the user’s touch-enabled device. This data is (optionally) preprocessed, then input to the neural network, and the network provides output which indicates the most likely match from the hand-drawn image, to known schematic symbols.

The preprocessing step refers to any manipulation of the image required to format it before being input to the network. This step would be needed in the case of a large-scale multi-user system, where the many types of devices that connect to the system through the client application do not all have the same screen resolution, color set, etc. The incoming data may need regularization steps to crop, filter, and resize the images for best algorithm performance.

“CircuitMagic”

Written By: Ed Pataky, Autodesk Inc.

Algorithm Training

To train a multi-layer neural network to be able to successfully perform classification on a set of hand-drawn schematic symbols, a training set of data is needed. This dataset includes multiple hand-drawings of each type of symbol, possibly in different orientations, or drawn in slightly different styles.

Creating a Training Set

As an example, consider the (partial) training set shown in Fig. 8 which was used in the preferred embodiment for this invention. A sheet of standard size printer paper was divided up into 10 rows and 8 columns (5 rows shown). Each column contains multiple hand drawings (each row) of the same class of schematic symbol. Each box represents a sample image to be used in algorithm training for which the class it belongs to is known. In this example, there are 8 different classes, representing the schematic symbols: resistor, capacitor, inductor, battery, independent voltage supply, independent current supply, ground, and an op-amp symbol.

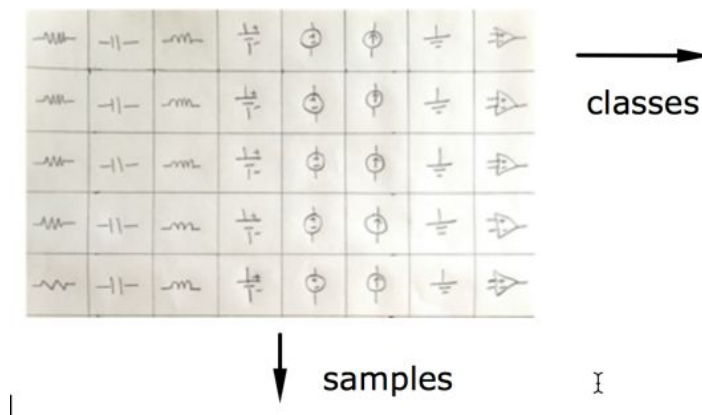


Figure 8: Training Data Set

“CircuitMagic”

Written By: Ed Pataky, Autodesk Inc.

In Fig. 9., we see a visualization of the pre-processing steps used to create the database of images used for algorithm training. In addition to formatting and regularizing the images, we need to create the ‘hints’ for the algorithm that will allow it to classify symbols and be immune to changes in rotation, mirroring/flipping, and scaling of the components. For this reason, as shown in he the figure, for each sample image in the original training set, several operations are performed.

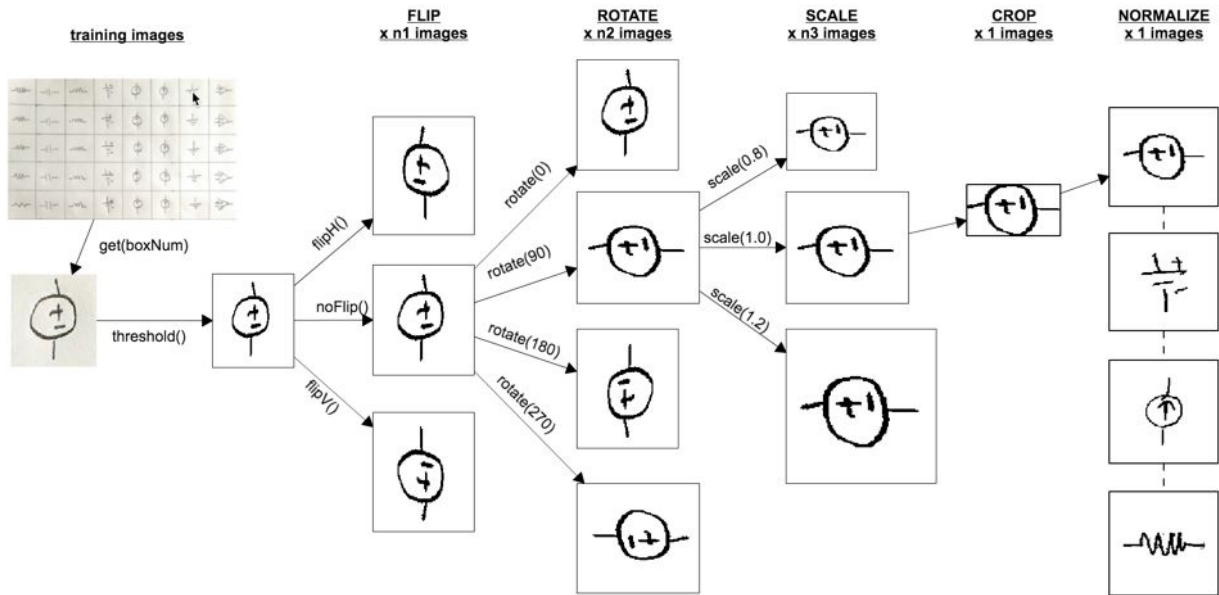


Figure 9: Image Pre-Processing For Training Data Set

First, the image is passed through a threshold detector, which converts each color image to a black and white image, such that any pixel with a combined color value (each of R,G,B values multiplied together) less than the threshold is considered black and set to pure black, and otherwise set to pure white, where we assume the maximum values for all R,G, and B give pure white. The end result is a huge reduction in image size, since, a color image with red, green and blue values for each pixel is converted to an image with only a binary value 0 or 1 for each pixel, indicating white or black.

To create several versions of each image, a version of the image not flipped, then flipped across the horizontal axis, and another across the vertical axis is created. Then for each of these images, a version of the image is created that is rotated by each rotation value of 0, 90, 180, and 270 degrees. For each of these rotated images, a scaled version is created with a ratio of 80%, 100% and 120% of the original size. Finally, the image is cropped to the extents of the min and max black pixels found in the image in both x and y directions.

Once all images are created for each sample image from the original set, a final step is performed where the maximum bounding box size of all images is found first, and then all images are placed in the center of a new blank image of this common size. This step

“CircuitMagic”

Written By: Ed Pataky, Autodesk Inc.

normalizes the images as if they were all drawn in an area of this size. Note that in the last step (not shown in the figure), after centering images, all images are also again scaled down by a fixed factor to reduce the number of pixels in each image. This can be done as far as possible as long as accuracy is acceptable in the final algorithm performance.

In all, if we consider the 8.5” x 11” page which was split up into 88 separate images, this preprocessing converts the original 88 image set into $88 \cdot n_1 \cdot n_2 \cdot n_3$ images. In the case of the preferred embodiment and example processing used here, $n_1=3$, $n_2=4$, $n_3=3$, and so the full training set was $88 \cdot 3 \cdot 4 \cdot 3 = 3168$ images. To enhance the training set, since the more samples the better for training, 5 identical copies of each images was used, which brings the total to 15840 images. The final resolution of the centered images from the last step was 116x116 pixels. Before inputting to the algorithm, the size was reduced by 50% for a size of 58x58 pixels, amounting to $p=3364$ pixels.

For the algorithm training, this means this one sheet of paper, and after the preprocessing noted above, produced 15840 samples with 3364 features each. The number of features should be much less than the number of samples, so the resizing steps help a great deal toward better convergence and performance of the algorithm. How many and what type of rotation angles to use, scale factors, flip/mirroring axes, resize factors, and all other variables are specific to each application, and need to be tuned for best performance.

Choosing h, the # of Hidden Layer Nodes

In literature, there is no standard way to choose the optimal number of hidden layer nodes (or the # of hidden layers for that matter) a priori. The only proven way is to run the network as many times as needed, while iterating through multiple possibilities, to arrive at the # of nodes that gives best results. For the preferred embodiment, from the training set, a dimensionality reduction technique known as Principle Component Analysis (PCA) was performed, such that the PCA algorithm will transform the dataset samples into the minimum number of features that can ‘explain’ X % of the variance between samples. This percentage was set to 95, and the number of hidden nodes was set to the number of features from the dataset that cover 95% of the variance between them, according to the PCA results. Any method can be used to choose the # of hidden layer nodes, the simplest being to iterate different values to find the minimum number that gives good performance. In this embodiment, 361 (19x19) hidden nodes were used in 1 hidden layer.

“CircuitMagic”

Written By: Ed Pataky, Autodesk Inc.

Batch Training The Neural Network

To train a neural network, the standard procedure is to:

1. Setup the network topology (# layers, # nodes, type of activation functions used, connections, etc). *In the preferred embodiment, there were $p=3364$ input nodes, plus one so-called bias node with the value of 1, 1 hidden layer with $h=361$ nodes where each performs a tanh function plus one bias node with a value of 1, and 24 output nodes. There are 24 nodes because the 8 symbols were considered different classes for each value of flip in [NONE, HORIZONTAL, VERTICAL]. In other words, anticipating that flip invariance would be difficult to train, a component that is flipped is considered a different class from one that is not.*
2. Initialize all connection weights with small random numbers
3. In a loop over all samples in random order:
 - a. Calculate “forward” by inputting the sample and calculating all hidden layer outputs
 - b. Calculate all outputs at the output nodes given the hidden layer outputs
 - c. Going “backward”, using an error function appropriate for the problem, calculate the change in hidden layer to output layer weights, and input layer to hidden layer weights using gradient descent
 - d. Update the input-hidden layer weights and the hidden-output layer weights
 - e. Check convergence criteria and exit if satisfied

In the preferred embodiment, the back-end software was written in python, and open-source machine learning libraries PyBrain (pybrain.org) and Scikit-Learn (scikit-learn.org) were used, which both use numpy (numpy.org) and scipy (scipy.org) python math libraries. Scikit-learn was used to format the data and run the PCA algorithm, and PyBrain was used for creating, training, and running the artificial neural network. The Python PIL library was used to read in and manipulate image data. Note that the NN training algorithm parameters included a learning rate $\eta=0.0001$, momentum=0.85, learning rate decay=0.95, weight decay=0.0001, validation proportion = 0.25. The Pybrain BackpropTrainer algorithm was used to train a FeedForwardNetwork type network, and the trainUntilConvergence function was used to perform the training. Using the Pybrain library, the trained neural network, including all model parameters, can be saved to an XML-formatted file for later use. The network can then be loaded from the file along with all the parameters as they were at the time training was completed, and the network can be “activated” on new samples to classify them.

“CircuitMagic”

Written By: Ed Pataky, Autodesk Inc.

Online or Live Method of Training The Neural Network

Note that algorithm training can be done entirely offline, and only the trained parameters are required at runtime. Training the algorithm this way would on one hand require that the system programmer create and feed in all the training samples, but on the other hand it would be ready to go before the first user of the system gives it a try. Another method of training would involve having the users help to create the training set.

If we consider that we have a system setup that can handle hundreds of thousands of users connecting to the cloud servers from the schematic application, we can note that the app designer can choose to involve the user in the training process. At prescribed intervals, or maybe only the first time the user uses the application (as part of a setup phase), every user of the system can be asked to provide input by hand-drawing certain schematic symbols, where they will mark each predicted classification result as correct or not correct. Using this information from thousands of users, on the back-end, the algorithm can be trained once there is enough data to perform training, or it can be re-trained and updated periodically as new data comes in. In this way the training process can be more of a live and ongoing process than a one time batch process performed before the system is live. This live updating of network parameters can prove to achieve much higher accuracy in algorithm performance, since having many more samples for each class of symbol will help train parameters to the correct values. It also helps with invariance to rotation, style, scale and other variants for each symbol, since data from many users will inevitably have variations that we to account for.

Using The Neural Network

Once we have a neural network that has been trained, we can input sample images and classify them. At this point the back-end software can be connected to the server software, and enabled to take image data input and provide class information output. We refer again to Fig 5, and note that the trained network algorithm E on the cloud server D is loaded from a file (or database) G at the start of the system being live (or is already in memory from the last training task) and the server software F is configured to accept incoming connections from the client application A. Application A is an EDA CAD system (as in Figures. 4, 6) capable of sending image data (Fig. 5 B) to the server D for processing. The data is (optionally) preprocessed and then input to the trained algorithm which will return the most likely classification match(es) for the data.

In the preferred embodiment, since classification of this sort will likely never be 100% accurate for all inputs, all matches in order of likelihood (highest first) are returned to the application, so that the application can display to the user which symbols it believes are the best matches to what they drew. The user then can make the final decision of which component to place in the schematic, or they can choose to use the application menus to choose in the case no math was found.



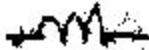
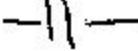



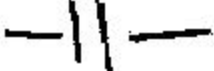




“CircuitMagic”

Written By: Ed Pataky, Autodesk Inc.

Results

Training error for training the network described above was 0%. Out of hundreds of samples, not even 1 image was misclassified, even considering rotated, scales, and flipped images.

The following example show the original image, plus the first 3 best matches as given by the algorithm after sending random image samples into the trained network. In fact, in all cases the correct best match was returned, returning the matching component to the original intended schematic symbol.

<i>Original Image</i>	<i>Best Match</i>	<i>2nd Best Match</i>	<i>3rd Best Match</i>
 (resistor)	 (resistor)	 (inductor)	 (capacitor)
 (inductor)	 (inductor)	 (resistor)	 (capacitor)
 (capacitor)	 (capacitor)	 (inductor)	 (resistor)