Detection of Hand Drawn Circuit Schematic Symbols

Aditi Kumar
Stanford University
aditik10@stanford.edu

Riyaz Merchant
Stanford University
ramercha@stanford.edu

***Abstract*—Digital transcription of hand drawn schematics offers engineers a powerful tool to quickly share and analyze circuit ideas in a fast paced innovative environment. The current state of the space does not offer a practical tool as of yet. Our goal is to create a classification CNN to identify components, and use another processing algorithm to attempt the counting of components in the hope of generating a bill of materials. Using a custom dataset, we trained a custom model, consisting of a pretrained MobileNet base, and custom neural network to act on the outputs of the MobileNet CNN to classify components. Dropout regularization and Adam optimizer were used during the training process. Component classification was highly successful, achieving accuracy of 96%. To achieve count, two primary algorithms were used, grid and sliding windows. However, these algorithms failed to classify components that were part of a larger schematic. Future work is needed to bring the classifier into a pragmatic tool.**

Introduction

Hand drawn schematics are a common graphical tool used in industry to brainstorm electrical designs. These hand drawn designs eventually have to be redrawn in a schematic capture software such as Altium or KiCad. As alumni of ME 218 we are familiar with the cumbersome process of transferring hand drawn schematics and the errors associated. Automating this process can help reduce human transcription errors as well as save time by removing the human from the process. Furthermore, this transcription could be built into mobile applications that can analyze a hand drawn schematic, speeding up development and improving brainstorming at the sketching stage.

The first key step in digitizing a schematic is to classify hand drawn symbols. These symbols are the graphical representation of true components that can be analyzed via software to test a circuit's efficacy. Therefore, component classification is a vital component of digital transcription.

In order to perform the classification, a neural network was used. Given the low volume of data available, the network was designed as a top layer to a pretrained model, MobileNet [1] CNN, which was trained using the Image-Net [2] dataset. We then designed a network that took the outputs of the MobileNet and classified components. The input to our model were 128x128 RGB images, which are first passed through the MobileNet CNN, then passed into the custom neural net, and output a softmax vector indicating the probability the input image belongs to the component class.

To reduce the scope of the problem, we focused on the seven common components we used during our ME 218 tenure. The components were: resistors, inductors, capacitors, MOSFETs, BJTs, diodes, and operational amplifiers. We also included a none category to handle things like text, wires, and power symbols. Each of the components had particular features we were hoping the model could use in order to classify successfully. For example, a diode has a triangular base, and a line above the vertex of the triangle.

In order to try and use classification to do some simple circuit analysis, we attempted to create a system that could scan a schematic, correctly identify components, and then produce a bill of materials. This would allow us to see if our classifier could be put to some analysis use, bringing it from a more theoretical exercise to a practical one.

I. Related Work

Several attempts have been made in the past to classify components in hand-drawn circuit schematic images. Bhattacharya et al [3] and Roy et al [4] proposed image segmentation techniques that did not involve deep learning for this process. In their work, Bhattacharya et al [3] devised a pipeline that comprised of image binarization, morphological closing of the circuit diagram, removal of wires using run length smoothing algorithm (RLSA) and localization of segmented components. They created an in-house dataset of 60 complete circuit diagrams with 14 classes of components. They achieved a localization accuracy of 91.28%. Their algorithm failed when components were drawn too close to each other and at nodal points and corners. Roy et al [4] used a combination of texture and shape based feature extraction along with a feature selection algorithm called ReliefF to rank attributes. These ranked features were then passed to different classifiers such as K-Nearest Neighbors and Naive Bayes to identify components in a dataset of 20 classes of individual component drawings. They achieved an average recognition accuracy of 93.83%. Their algorithm tended to misclassify components that were similar in overall shape or outer boundary.

Deep learning based methods for hand-drawn circuit component identification have also been implemented in the past. Wang et al [5] proposed a hand-drawn electronic component recognition method using a CNN and a softmax classifier. A sparse auto-encoder fed into the CNN which consisted of convolutional layers, a ReLU activation function and a pooling

layer. A hand-drawn dataset of 64 images of resistors, capacitors and diodes (3 classes) was created and then augmented by rotating the images. They were able to achieve 95% accuracy. Dey et al [6] also proposed a CNN-based method for circuit component recognition. They came up with a two-stage CNN model where the first stage was used to group together similar-looking components and the second stage was used to classify each component group into its actual classes. The method was evaluated on an in-house hand-drawn circuit component dataset with 20 classes of components, and it was found to have an accuracy of 97.33%.

Gunay et al [7] performed a survey of the three most common methods used for hand-drawn circuit and component classification which were namely, Support Vector Machine (SVM), K-nearest neighbor (KNN) and CNN. They implemented a CNN-based classification model on a dataset of four different hand-drawn components and concluded that CNNs are the best method for classifying circuit components. They achieved an average accuracy of 93.75% with their 12-layer CNN.

From past work, it is clear that a CNN-based approach involving multiple stages is the most accurate method of classifying circuit components. It is worth noting that deep learning methods have not been used to classify and count components in a full circuit schematic. This was only done using image segmentation and it required a run length smoothing algorithm for deleting wires between components. Due to the lack of availability of open-source hand-drawn circuit datasets, all previous work has involved dataset generation and augmentation.

II. Dataset and Features

To train and test a neural network for component classification, we created our own dataset of hand-drawn circuit components and used data augmentation techniques to increase the number of images. We selected 7 commonly observed components - resistors, capacitors, inductors, diodes, operational amplifiers, BJTs, MOSFETs - and drew 20 symbols of each component with slight adjustments to account for the variability in different people's drawings. We also drew 20 symbols that did not represent any of these but are commonly seen on circuit schematics such as batteries, power and ground, and straight lines representing wires. All of these images were drawn with a black pen on white paper to reduce preprocessing steps for grayscaling. Blank images of a plain white background were also included in this none class as plain white space does not represent any circuit component. For each class of components, we also added an image of a computer-generated schematic symbol. All images were resized to 120x160 pixels. In total, we had 168 raw images of components that were augmented by rotating seven times in increments of 45 degrees, passing through a Gaussian blur filter and an average blur filter, and skewing with an affine transformation to create 1760 images in total. The dataset was then divided into 80% training (1408 images), and 20% test (352 images). The scikit-learn [8] library was used for splitting the data and each class in both the training and test sets had roughly equal numbers of images. Figure 1 shows some sample images from the raw dataset.

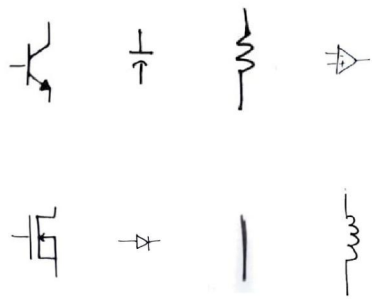

Fig. 1. Example Component Images in Dataset

For the second part of the project, the trained neural network was used to convolve over an image of a full circuit to identify components in the image and determine how many instances of each component are present. To create the dataset for this part, we collected our own and our friends' hand-drawn full circuit drawings. We limited these drawings to the ones made for the ME 218, Mechatronics, classes at Stanford to ensure all components in the drawings match with the components the network has trained on. A total of 15 images were generated by hand-drawing. No augmentation was performed and the images were not processed to have the same size. This is because we wanted our algorithm to be robust enough to work with full circuit schematics of any size. Figure 2 shows a sample full circuit image from this dataset.

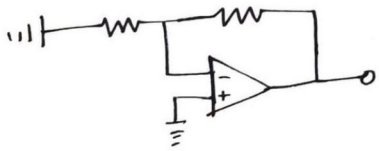

Fig. 2. Example Full Circuit Image in Dataset

III. Methods

A. Modeling

The model used to perform the classification is naturally divided into two parts, a pretrained MobileNet CNN, the base model, and a custom neural network whose input is the output of the MobileNet model, the top model. These models are combined to create an end to end model that accepts a 128x128 RGB image of a hand drawn component and then outputs the classification probabilities. RGB was used in order for us to leverage the pretrained models.

All model construction was implemented using the TensorFlow library [9]. We accessed the MobileNet model through

the keras application class of TensorFlow, and configured the model to omit the top classification layer, since we attached our own model instead. We selected the weights that result from training MobileNet on the ImageNet dataset. We selected average pooling between the convolution layers, in hopes this lead to weights that did not focus on edges or other sharp features. This part is solely designed to leverage a pretrained model for feature extraction. The top model, which training is performed on, has one hidden layer of 512 neurons with a hyperbolic tangent activation, and an 8 neuron output layer with a softmax activation. The hidden layer adds enough model complexity to help classify the MobileNet outputs. The hidden layer provided the model more weights to train, and thus helps with performance. The neuron count of 512 was chosen to halve the output from the fully connected 1024 layer on the output of the MobileNet. We chose to use hyperbolic tangent given its normalizing properties. While S-shaped like sigmoid, tanh is centered around 0, thus the mean of the outputs from the hidden layer are more likely to center around 0. Furthermore, during training, strong negative inputs will result in a -1 output, not 0 like sigmoid, allowing for faster weight updates, preventing the network from getting stuck during training. Given the use of hyperbolic tangent, we used Xavier initialization [10] to prevent instability from weight initialization.

The final output layer provides the probabilistic categorization of the features. The layer consists of 8 neurons, one per class, with a softmax activation. The softmax activation allows the model to output the probability that the component belongs to each class, in other words:

$$p(y = i|\eta) = \frac{e^{\eta_i}}{\Sigma_{j=1}^{k} e^{\eta_i}} \tag{1}$$

where η_i represents the components of the vector outputted by the layer.

B. Training and Optimization

We trained the top model using the built in fit function supplied by TensorFlow [9]. As state, we only performed training on the top model. This was achieved by first passing the dataset images through the base model, and using those outputs to train the top model. We chose to use an Adam optimizer with a learning rate of 0.001, with the default β and ϵ values. Given we were using a softmax output, we opted for a categorical cross entropy loss function to optimize around.

$$CE = -\log(\frac{e^{\eta_i}}{\Sigma_{j=1}^{k} e^{\eta_i}}) \tag{2}$$

Equation 2 is written in its simplified version given the true y value is a one-hot vector, thus the sum reduces to a singe component when $y = i$. As mentioned before, we used Adam to perform the optimizations, letting TensorFlow perform the process for us. We chose to use Adam because of its benefits when dealing with sparse and noisy gradients. Since Adam uses momentum to update the learning rate per parameter, customized learning rate can be used, allowing all parameters to update at a rate that is consistent with every other parameter, unlike SGD which uses the same learning rate for all parameters, which can cause training to be longer as some parameters update quicker than others [11]. Furthermore, Adam performs better on noisy or sparse gradients. Given our small dataset, we thought this might lead to a weak training signal, and thus concluded Adam would be the better overall choice.

Due to the small dataset, we were concerned the model would overfit. We thus chose to add a 20% dropout layer on the hidden layer. This dropout layer effectively set the weights of 20% of the nodes to 0, removing them from the calculation. By dropping out randomly, each train step will see a slightly different layer, introducing noise into the training process. The dropout rate was a hyperparameter that was tuned during development.

IV. Experiments/Results/Discussion

A. Success Metrics

The success of the neural network was determined by its accuracy. Accuracy was defined as the proportion of correctly classified images to the total number of images classified. An image is correctly classified if the predicted label matches the predefined label for that image. Equation 3 below mathematically defines accuracy as the number of examples where the predicted label matches the true label over all examples.

$$accuracy = \frac{\sum_{i=1}^{n} 1\{\hat{y} = y\}}{n} \tag{3}$$

In the first part of the project, images of individual components were passed into the CNN for classification. Accuracy was based on prediction of labels for each component image. In the second part of the project, the trained CNN was convolved over a circuit schematic that was divided into smaller images either with a fixed grid or a sliding window. In this case, accuracy was measured by the prediction of the CNN on each smaller image. Due to the spacing of components in the circuit schematics and the grid size selected, the smaller images were divided in such a way that not more than one component was present in each small image.

B. Hyperparameter Tuning

There were several hyperparameters that we tuned during experimentation. The list includes: learning rate, regularization, batch size, base model type, neuron number, and activation functions. Most were selected based on how the model performed on the test set with said values. Others were based on the recommended values found in the literature or TensorFlow documentation. If the model performed well with these starting values, they were used.

The training parameters we experimented with were learning rate, batch size, and epoch number. We began with a learning rate of 0.001 and the default beta and epsilon values for the Adam Optimizer. These were the recommended settings from the literature [11]. We experimented with the learning rate to see how increasing or decreasing would affect training.

Increasing made training highly unstable, with the loss and accuracy varying widely between epochs. Decreasing made no significant difference in final trained accuracy and just slowed the process down. Sizing the batches was also tuned. We found using 64 batches and running enough steps to cover the dataset per epoch worked the best. The smaller batches provided a good estimate, and helped make training faster by applying more steps per epoch with the smaller batch. Epoch count was determined experimentally as well. We found 10 epochs worked the best.

The next, and arguably most critical, parameter was the base model selection. We started with Inception V3 as it provided the best overall performance as outlined in the TensorFlow documentation [9]. While the model trained well with this base, but it tested at roughly 60% accuracy on the test set. After reviewing the literature, we found the MobileNet model may suit our needs better [1]. Given there were fewer parameters and convolution layers, we felt perhaps this would not overfit as much as Inception V3, giving our model more flexibility during training. After switching to this model, we saw a test accuracy of 86%. This was a dramatic improvement.

C. Regularization

Given the small size of our dataset, overfitting was a major concern. Preliminary testing showed our model training to near 100% but the test was not breaking 90%. We thus decided to experiment with regularization to see if we could improve our test accuracy. Table I summarizes the accuracy results. We first assumed a Gaussian prior across our weights, given we had a large network layer of 512 neurons. Thus, we started with L-2 normalization and this did help. However, training was more difficult and results were not consistent. We then simply tried stopping the training early. This also helped, but did not perform as well as L-2. We then took a different approach and tried to use dropout instead. We started with the 20% default, and that worked well. We then proceeded to decrease the rate to 10%, but discovered this worsened performance.

TABLE I
REGULARIZATION TECHNIQUES

Regularization Technique	Accuracy
L2-Norm	93.521%
10% Dropout	94.92%
20% Dropout	96.34%
Early Stopping	92.13%

D. Results and Discussion

Training loss and accuracy of the CNN on the individual component images was recorded over 15 epochs and is shown in Figure 3. As training takes place, the loss decreases marginally and flattens out after about 3 epochs, and the accuracy of classification increases from 96% initially to 99.99% after 15 epochs. The accuracy curve also flattens out at about 3 epochs.

Using the optimally tuned hyperparameters, our model achieved a detection accuracy of 96% on the test set. This is

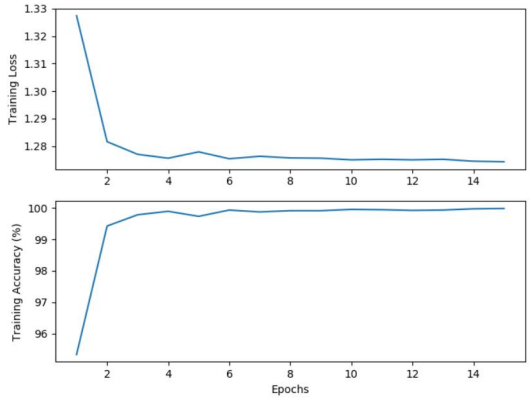

Fig. 3. Training Loss and Accuracy

comparable to the experiments run by Dey et al [6]. Figure 4 shows the confusion results. In general the model did very well, and in multiple classes, suffered no confusion. The classes that fared the best were capacitors, MOSFETs, and none. The class that fared the worst was diodes, getting confused for MOSFETs and Op-Amps. All three components have a major triangular feature which would explain the confusion. Resistors and inductors were also confused for each other. These components have similar shape, especially when hand drawn, explaining the confusion. The model confused some BJTs for MOSFETs, predicting the latter more accurately. This could be explained by the increased complexity of the MOSFET symbols, allowing for better feature recognition by the model.

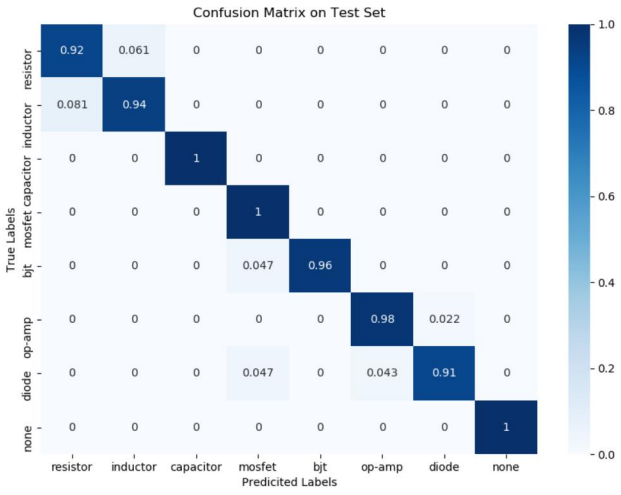

Fig. 4. Confusion Matrix

Grid Method: The model was used to scan over full circuit schematics using two different techniques. The first technique, the grid method, involved creating an grid on top of the schematic images and then classifying each grid pane, which was designed to be 128x128, separately using the model. The

circuit shown in Figure 5 was divided into smaller images using the grid method and classified with the CNN to generate the heatmap shown in Figure 6. The heatmap shows probabilities assigned by the CNN to each rectangle of the grid. In this case, it can be seen that the CNN predicted capacitor with a high probably for the rectangle containing the right hand side capacitor, but failed to identify the capacitor on the left side of the circuit. On the other hand, the CNN classified all the white space in the image as none with very high probability, but seemed to fail around the power and ground symbols. Power and ground symbols were also included in the training and test sets for the none class.

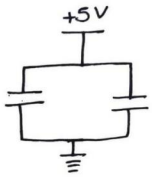

Fig. 5. Example Circuit

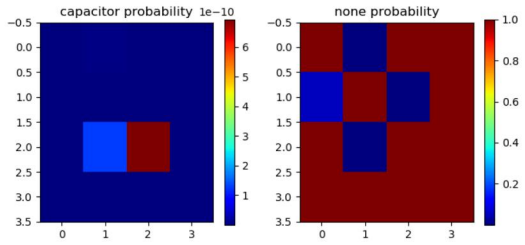

Fig. 6. Heatmap for Grid Method

Sliding Window Method: The second technique, sliding windows, classified the various components in the circuit by moving a window of fixed size through the rows and columns of the full circuit schematic. Figure 7 shows the results of classification using the sliding window method on the circuit in Figure 5. This method was unable to detect the capacitors in the circuit as the heatmap shows the very low probabilities with which capacitors were predicted on the image. However, this method performed better in classifying the white spaces in the circuit as none which is illustrated by the red colored pixels all around the edges of the heatmap.

The sliding window approach was very time-consuming and did not offer any improvement in accuracy as compared to the grid method. As small windows were created and slid over each full circuit schematic, many windows were created where only a small part of the component was visible. Due to this, the accuracy of this method was lower than that of the grid method as it was more difficult to classify the small parts of the components visible in the windows.

The grid approach was computationally faster than the sliding window approach and generally gave more accurate

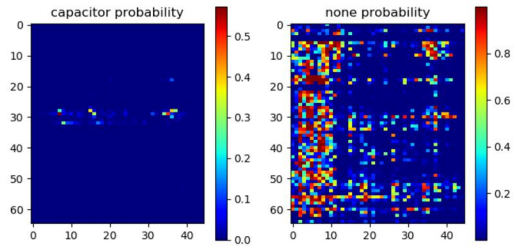

Fig. 7. Heatmap for Window Method

results. However, the accuracy of this approach was also poor. Since the none class contained power symbols, wires, blank space and nodes, this was the most common guess for most of the schematic, and many times components were misclassified as none objects.

V. Conclusion

This paper proposed a method for identifying circuit components in hand-drawn schematics using a neural network built on top of the MobileNet CNN. A dataset of 1760 images of individual circuit components was generated by hand-drawing 8 classes of circuit components and then performing various methods of data augmentation. Using 20% dropout regularization, the CNN was able to achieve 96% accuracy in classifying the test set of individual component images. This CNN was then used to classify components in hand-drawn circuit schematics using two different approaches - grid and sliding windows. The grid method performed marginally better than the sliding windows method and it was computationally faster.

Given more time, the critical issue we want to explore is trying component recognition in the presence of wires. As discussed, both grid and sliding windows performed poorly, even though the classifier worked well on individual components. In order to devise a practical use of the classifier, this needs to be explored further. Perhaps the fist step would be to create an algorithm that can scan a full schematic and recognize any component as a feature, extract the component, and run it through the classifier. With more computational resources and data, we could have designed our own custom CNN and trained it to extract features that are essential to schematic component. We would also like to expand the component set to account for a larger variety of digital and analog symbols as well.

Acknowledgments

Thank you to Elizabeth Vasquez, Rianna Jitosho, Maqdooda Merchant, Adnan Merchant, and Sophie Merchant for drawing circuits for us to expand our dataset. Thank you Ian Tullis for helping us through the project, offering advice for next steps.

CONTRIBUTIONS

Aditi: Handled the generation of the dataset. This included providing the proper augmentation and splitting images into train and test sets for use in modeling. Wrote all needed scripts to perform the data augmentation.

Riyaz: Handled TensorFlow infrastructure. Generated the scripts to train and test the combined model and perform the grid and sliding window classification on whole schematics.

Both team members brainstormed ideas to try and improve classification. Given Riyaz's past experience with TensorFlow, implementation through him was most efficient. However, team members met and coded together.

REFERENCES

[1] Andrew G. Howard et al. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. cite arxiv:1704.04861. 2017. URL: http://arxiv.org/abs/1704.04861.

[2] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[3] Archan Bhattacharya et al. "Circuit Component Detection in Offline Handdrawn Electrical/Electronic Circuit Diagram". In: *2020 IEEE Calcutta Conference (CALCON)*. 2020, pp. 80–84. DOI: 10.1109/CALCON49167.2020.9106527.

[4] Soham Roy et al. "Offline hand-drawn circuit component recognition using texture and shape-based features". In: *Multimedia Tools and Applications* (2020), pp. 1–21.

[5] Haiyan Wang, Tianhong Pan, and Mian Khuram Ahsan. "Hand-Drawn Electronic Component Recognition Using Deep Learning Algorithm". In: *Int. J. Comput. Appl. Technol.* 62.1 (Jan. 2020), pp. 13–19. ISSN: 0952-8091. DOI: 10.1504/ijcat.2020.103905. URL: https://doi.org/10.1504/ijcat.2020.103905.

[6] Mrityunjoy Dey et al. "A two-stage CNN-based hand-drawn electrical and electronic circuit component recognition system". In: *Neural Computing and Applications* (2021), pp. 1–24.

[7] Mihriban Günay and Murat Köseoğlu. "Classification of Hand-Drawn Circuit Components by Considering the Analysis of Current Methods". In: *2020 4th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*. 2020, pp. 1–5. DOI: 10.1109/ISMSIT50672.2020.9255047.

[8] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[9] Martın Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[10] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*. 2010.

[11] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. URL: http://arxiv.org/abs/1412.6980.