Handwritten Equation Solver Using CNN

Chase Clark

In this project, I created a CNN using Python that can recognize math symbols in images. My project breaks down into three main phases or sections. Data preprocessing, model creation, and testing of the model. In this document I will explain how I approached each section and what challenge I faced along the way.

I found my dataset on a popular data science site called Kaggle. There, I found a dataset that includes handwritten numbers and symbols in image format. After extracting each symbol into their own subfolder I was ready to start processing them into format that is easier to work. Next, I created a method that loops through every image in the subfolder and extracts the important features. Before working with the images, I inverted the colors so it is a black background with a white foreground. This supposedly increases the accuracy of the opencv library I was using. Then for each image, I used opencv to identify contours and create bounding rectangles around them. Next, I used the dimensions of the bounding rectangle to crop the image down to a normalized size of 28 x 28. Finally, I flatten the image and store the data inside a csv file and append the label in the final column. Now the data is ready to be used for training.

Creating the model was the hardest part of the project for me. I had to research and find examples on the web before I was able to put together an effective model.( Reference: Vipul Gupta <https://medium.com/@vipul.gupta73921/handwritten-equation-solver-using-convolutional-neural-network-a44acc0bd9f8> ) Before creating the model, I looped through my csv file and reshaped the data back to the original (1,28,28) format. Then, using keras, I added the first layer to the model which is Conv2D. This layer takes in the input image which is of size 28 width, 28 height, and 1 channel depth. I set the amount of output filters to 30 and the kernel size to 5 by 5. The next layer is a MaxPooling2d layer with a pool size of 2 by 2. This reduces the inputs by half which summarizes the features we are looking for. Next, I add a another Conv2d followed by a second MaxPooling2d, to further reduce the inputs. Next, we add 2 simple layers to prepare our data for the Dense layers. We use Dropout(0.2) to drop 20 percent of inputs which helps prevent overfitting. Then, we add a Flatten layer which prepares our data to be processed by the upcoming Dense layers. Finally, we add in three consecutive Dense layers which have smaller and smaller outputs. The final Dense layer has the exact amount of outputs to match all of symbols we are training for. In my case, this is number is 12(0-9,+,-). After compiling and fitting the model to my training data, I save it to a JSON file for easy use in testing. I ran some metrics while training my model and it had around a 98% percent accuracy score.

For testing my model, I used Microsoft Paint to draw some equations. Then, in my Python file, I imported my model.json and the corresponding model.h5 weights. Next, I load in my test image and run a processing technique similar to what I did in the preprocessing phase. Once, I have isolated all the symbols from the test image, I use my model to perform a prediction and give me the result. Finally, using the result, I can store it in a string equation variable and use Python’s eval() function to evaluate the result.

Overall, the model performs well. The only issue I saw was it confused the numbers 2 and 7. In the future, I would like to add more symbols to my model and maybe create a front end application to run it.