**Capstone Project User Guide:**

**Using Machine Learning to Perform Handwriting Recognition on Mathematical Equations**

Brendan Turner

Grand Canyon University

December 15, 2021

**General Information**

This project aims to identify and solve images of handwritten mathematical equations. Successful implementation of this project will build a model that will take any image of a mathematical equation, whether on paper or drawn on a digital sketchpad, and identify each of the individual characters contained in that image. It uses a classifier that will recognize what each character is, and turn these results into an equation string. This string will then be solved by a custom-built function, and the solution is displayed below the input image for the user.

**System Summary**

The program is written in Python 3.6, and the Jupyter Notebook was used to write and test the code. There are two main files: one for designing, running, and saving the image classifier, and a main.py file, which runs the GUI and performs the main functionality of the program. This entails gathering the input image, segmenting the characters of the image, running each segmented character through the classifier, and using the results to put together the equation string to be solved. The main program uses PySimpleGUI as the user interaction model, OpenCV and imutils to perform segmentation, and SymPy to perform the equation solver. The classifier’s libraries include NumPy and Pandas for data handling, Pillow for image extraction, and Keras through TensorFlow to build the model itself. All of these libraries must be installed prior to running the files.

**Getting Started**

This application is very simple in its nature, as since it is designed for one purpose, it only has one user function. This is to click on the “Choose File” button to pick which image you’d like to upload (future versions may include a sketchpad to draw your equation in the app). The system will take care of everything else

**Using the System & Troubleshooting**

The main benefit of the simplicity of this app is that beyond uploading an image, there is nothing else that the user needs to do. The program will analyze the image itself and produce a solution in the window beneath the image itself. This image that you upload must be either a JPEG or PNG file. If you choose an incompatible image, or one that does not display a mathematical equation, you will get an error message and be returned to the main screen.

**Help and Contact Details**

If you have any questions about this product, the developer can be reached at:  
 [brendanrturner1@gmail.com](mailto:brendanrturner1@gmail.com) or by phone at 908-665-8056.

**FAQ**

The data sourced for this project comes from the HASYv2 dataset (original document found [here](https://arxiv.org/pdf/1701.08380.pdf)). This was inspired by the famous MNIST and expounds upon it by including myriad handwritten mathematical LaTeX symbols (letters, numbers, operators, set notation, calculus symbols, etc.) and general extras, like male/female or horoscope symbology. This was chosen for its ease of access and the fact that the images are already in 32x32 pixel PNG formats, unlike the InkML of the CROHME dataset which is very complex to extract from.

To compensate for the disparity in class size for certain symbols, the model performs under/over-sampling and image augmentation on all characters. Each is stretched, rotated, zoomed in & out, shifted and distorted slightly so as to manufacture numerous new data to train the classifier on.

As for how the model actually classifies these images, it uses a Deep Convolutional Neural Network. The general methodology of the CNN is as follows. This is a neural network (meaning it is designed to mimic how humans learn patterns) composed of several layers. After taking in the image it will classify, the network looks at all possible groups of pixels (in this case, 5x5) to try find patterns in the pictures, which is also sometimes referred to as “subsampling”. These subsections are compared at both individually and with their neighbors to piece back together the image and find connections. These connections serve as nodes in the hidden layers of the network. The first few layers can identify lines, curves, indents, and colors by looking at these overlapping features. Further layers in the network can build on these existing pieces of information to recognize gradually more complex features. When it reaches the end of the network, as with any CNN, class scores are computed. This is a calculated probability that the image belongs to each of the predefined set of classes. Whichever is the highest is what the network assigns the image to be in.

The way that the hidden layers of the network actually carry out this detection is as follows. Each layer of a CNN transforms one group of activations (which choose to send information in direction X or direction Y) to another through a function. We use four main types of layers to build CNN’s: Convolutional, Pooling, ReLU, and Fully-Connected. We stack these layers to form a full architecture. For example, the input layer might contain the raw pixel values of an image (its height, width, and greyscale color) while the convolutional layer will compute the output of neurons that are connected to the local subsections of the input image, each computing a dot product between their weights and a small region they are connected to in the input volume.