Smart Math Canvas: Interactive Equation Detection and Solution Prediction

Faris Bin Asif
BS in Computer Science
Fast National University
Karachi, Pakistan
k200270@nu.edu.pk

Mubin Saeed
BS in Computer Science
Fast National University
Karachi, Pakistan
K200211@nu.edu.pk

Arham Nasir
BS in Computer Science
Fast National University
Karachi, Pakistan
k200195@nu.edu.pk

***Abstract*—The Air-Writing Math Recognition and Calculation System is an innovative project that leverages OpenCV and a Convolutional Neural Network (CNN) model to recognize and solve mathematical equations written in the air.**

I. Introduction

In recent years, there has been a growing interest in the development of interactive and intuitive systems that enhance the way we interact with computers. Hand gesture recognition, computer vision, and machine learning have played significant roles in revolutionizing human-computer interaction. In this project, we present an innovative application of these technologies by creating an Air-Writing Math Recognition and Calculation System, leveraging the power of OpenCV and a Convolutional Neural Network (CNN) model.

This project seeks to replace traditional pen and paper with a novel and interesting method for users to enter and solve mathematical equations. Users can generate basic mathematical expressions by just writing them in the air; these expressions are instantaneously recognised when you move your pointer to a calculate button and shown in a separate paint window. The system is additionally made to identify equations written in the air and calculate their answers.

The project utilizes the OpenCV library, which is widely recognized for its computer vision capabilities. OpenCV provides a rich set of functions and algorithms for image processing, object detection, and tracking. By leveraging these functionalities, we can detect and extract the air-written equations in real-time, enabling seamless interaction between the user and the system.

II. Why we used Convolutional Neural Networks

To recognize the mathematical symbols and operators, we employ a Convolutional Neural Network (CNN) model. CNNs have proven to be highly effective in image classification tasks and have been extensively used in various computer vision applications. The CNN model is trained on a large dataset of mathematical symbols and operators, enabling it to accurately classify the handwritten expressions extracted from the air.

Identify applicable funding agency here. If none, delete this.

A. Ability to Learn Hierarchical Features

CNNs excel at automatically learning hierarchical features from input images. In the case of mathematical equations, these hierarchical features can represent different components such as numbers, operators, and symbols. By learning these features at various levels of abstraction, CNNs can effectively classify and recognize the individual elements of the equations.

B. Robustness to Noise and Variability

Handwritten equations in the air may contain inherent noise, variations in writing styles, and inconsistencies in stroke thickness. CNNs have demonstrated robustness to such variations, enabling them to effectively handle the inherent challenges of recognizing handwritten equations. This robustness ensures accurate classification and recognition of the equations despite variations in writing styles and other potential sources of noise.

C. High Performance in Image Classification

CNNs have achieved remarkable success in image classification tasks, including handwritten digit recognition, object detection, and character recognition. The underlying architecture of CNNs, with their convolutional and pooling layers, enables them to capture and learn intricate features present in images. This performance in image classification makes CNNs a natural choice for recognizing and classifying handwritten equations in the Air-Writing Math Recognition and Calculation System.

III. Workflow

The workflow of the project involves several steps that enable the system to interpret the air-written equations and provide real-time answers to the user.

The process begins with the user writing mathematical expressions in the air using pointer in the air. The first step in the workflow involves using computer vision techniques, specifically those provided by the OpenCV library, to detect and extract the handwritten equations from the background. This process involves masking out the user's pointer (must be blue).

Once the equations are written onto the paint window, the screenshot is taken and passed to a trained CNN model. The CNN model has been trained on a large dataset of labeled mathematical operators and numbers, allowing it to recognize and classify the individual components of the equations. The CNN architecture, with its convolutional and pooling layers, enables it to learn intricate features present in the images and make accurate classifications. By leveraging the hierarchical nature of CNNs, the model can identify numbers and operators within the equations. The visual feedback allows the user to see the equations they have written in the air and verifies that the system has interpreted their pointer.

Furthermore, the system performs the final step of calculating the solution to the recognized expressions. By evaluating the expressions, the system provides the answers to the mathematical problems. This functionality enables users to perform basic mathematical calculations using the air-writing interface, enhancing the interactive and intuitive nature of the system.

Overall, the workflow of the Air-Writing Math Recognition and Calculation System combines computer vision techniques, such as equation extraction using OpenCV, with the power of CNNs for classification. The system goes beyond recognition and also evaluates the equations to calculate their solutions. By leveraging the strengths of CNNs in image classification tasks, this project offers an innovative and engaging way to interact with mathematical equations using air-writing gestures.

IV. Methodology

In this project we have used openCV for the work where we have to input what user has written into the canvas and take screenshot of it in the directory. And then the trained model uses bounding box and classification to show and predict the expression.

A. OpenCV

Firstly, the code sets up trackbars using the OpenCV function cv2.createTrackbar(). These trackbars allow the user to adjust the upper and lower values of hue, saturation, and value, which will be used to detect a specific color range.

Next, the code initializes arrays to handle different color points. In this case, it uses a single array called bpoints with a maximum length of 1024. The code also initializes an index variable black index for marking points of a specific color. A kernel is created using the NumPy library for morphological operations such as dilation. A list of colors and a color index variable are defined to represent different colors. These colors are used to draw rectangles and labels in the paint window. The code creates a blank canvas using NumPy with dimensions 471x636 and a white background. The height and width of the canvas are stored in variables. Rectangles are drawn on the paint window using the cv2.rectangle() function, and labels are added using cv2.putText() to indicate the functionality of each rectangle. The screen and window dimensions are set to define the size of the GUI window using the screen width, screen height, window width, and window height variables. The window is made resizable by calling cv2.namedWindow() with the cv2.WINDOW NORMAL flag. Then we initialized the webcam by creating a VideoCapture object using cv2.VideoCapture(0).

Inside the main loop, the frame is read from the webcam using cap.read(), and the frame is flipped horizontally using cv2.flip() to mimic a mirror effect. The frame is then converted from BGR to HSV color space using cv2.cvtColor() to facilitate color detection based on hue, saturation, and value. The trackbar values are retrieved in real-time using cv2.getTrackbarPos() and stored in variables for upper and lower hue, saturation, and value. Buttons representing colors are drawn on the frame, and labels are added to indicate their functionality. A mask is created by applying the upper and lower HSV values to the frame using cv2.inRange(). Morphological operations such as erosion, opening, and dilation are performed on the mask to enhance the detection. Contours are found in the mask using cv2.findContours(), and the largest contour is selected based on area. The minimum enclosing circle is drawn around the contour, and its center is calculated using moments. If the detected center is within the region of the buttons, various actions are performed based on the button's position. For example, if the center is within the "CLEAR" button, the canvas is cleared by resetting the bpoints array and setting the paint window to white. If the center is within the "BLACK" button, the color index is set to black. If the center is within the "CALCULATE" button, a screenshot of the paint window is taken, saved to a file, and additional functions are called. If the detected center is outside the button region and the color index is black, the center is appended to the bpoints array. If no contours are found, a new deque is added to the bpoints array to avoid errors. Lines are drawn on both the frame and the paint window based on the stored points in the bpoints array, with each color represented by a different index. The resulting frames, paint window, and mask are displayed using cv2.imshow(). The code continuously checks for the 'z' key press, and if detected, the application is stopped by breaking out of the loop.

B. Model

The dataset used for training the model consisted of grayscale images of handwritten characters. To enhance the performance of the model, we employed an image data generator from the Keras library. The generator was configured with various augmentation techniques, including rescaling, shearing, and zooming, to improve the model's ability to generalize and handle variations in the input data. Additionally, a validation split of 25 percent was applied to evaluate the model's performance during training. The training data, located in the 'CompleteImages' directory, was loaded using the flow from directory() function provided by the image data generator. The images were resized to a uniform size of 40x40 pixels and converted to grayscale to simplify the input representation. The data was divided into batches of 32 samples for efficient training. The labels were encoded using a categorical format, enabling the model to perform multi-class classification. The training and validation sets were

shuffled to ensure randomness, and a fixed seed value was used to maintain reproducibility. For the model architecture, we designed a convolutional neural network (CNN) using the Keras Sequential API. The network consisted of multiple layers, starting with a 2D convolutional layer with 32 filters and a 3x3 kernel, followed by a rectified linear unit (ReLU) activation function. Max pooling was applied with a 2x2 pool size to reduce the spatial dimensions. Another convolutional layer with 64 filters and a 3x3 kernel was added, followed by max pooling. The output was flattened and passed through two fully connected (dense) layers with ReLU activation. The final dense layer consisted of 16 neurons with a softmax activation function, providing the probabilities for each class. The model was compiled using the Adam optimizer with a learning rate of 5e-4 and the categorical cross-entropy loss function. We also monitored the accuracy metric during training to evaluate the model's performance. To train the model, we used the fit() function, specifying the training and validation sets, and performed two epochs of training. The training progress was displayed to monitor the loss and accuracy improvements over time. Once the training was completed, we saved the model's learned weights in the 'my model weights.h5' file. Additionally, we stored the model's architecture in JSON format using the to json() function and saved it in the 'my model architecture.json' file. The methodology involved applying image data augmentation techniques, loading the training and validation datasets, designing a CNN model, compiling and training the model, and saving the trained model's weights and architecture. These steps allowed us to develop an effective deep learning model for the classification of handwritten character images, demonstrating its potential for various applications, such as optical character recognition and digitization of handwritten content.

Firstly, a list of character labels was defined, representing the characters to be recognized. The labels included symbols such as '%', '*', '+', '-', and digits from '0' to '9', as well as brackets '[' and ']'. These labels served as the classification categories for the recognition system. Next, a representative image of a handwritten mathematical expression was selected for testing and evaluation. The image was loaded using the OpenCV library and stored in the 'image' variable. To perform character recognition, a prediction function was implemented. The function took an input image and utilized the trained CNN model to predict the label of the character present in the image. The image was resized to a standard size of 40x40 pixels and normalized to enhance the contrast and make the features more distinguishable. The normalized image was then fed into the CNN model, and the predicted label was obtained by selecting the class with the highest probability. The trained CNN model was then loaded from previously saved model architecture and weights files. The model architecture file, 'my model architecture.json', contained the network structure, while the model weights file, 'my model weights.h5', stored the learned parameters of the model. In the main function, the image was preprocessed to enhance the character edges. This involved converting the image to grayscale, applying Gaussian blurring for noise reduction, and performing edge detection using the Canny algorithm. The resulting contours were extracted, sorted from left to right, and filtered based on their size to eliminate small and large regions. For each retained contour, the bounding box coordinates were computed. The region of interest (ROI) within the bounding box was extracted from the grayscale image. This ROI was passed to the prediction function, which returned the predicted label for the character. To visualize the recognition results, the bounding boxes were drawn on the original image using green color. The annotated image was then displayed using the matplotlib library and saved as 'bounding-box-image.png'. Another function, 'function_other()', was implemented to convert the predicted character labels into a readable equation string. The function iterated over the collected character predictions stored in the 'chars' list, mapping each label to the corresponding character symbol using the predefined label-to-character mapping. The resulting equation string was printed as the output. After processing each character in the image, the 'chars' list was cleared to prepare for the next image.

V. Links

- Model training code
- Air Canvas Source code