Air-Writing Recognition

Michelle Jin. Crystal Lin. { mjin8, crklin } @ ucsc.edu

Baskin Engineering UC SANTA CRUZ

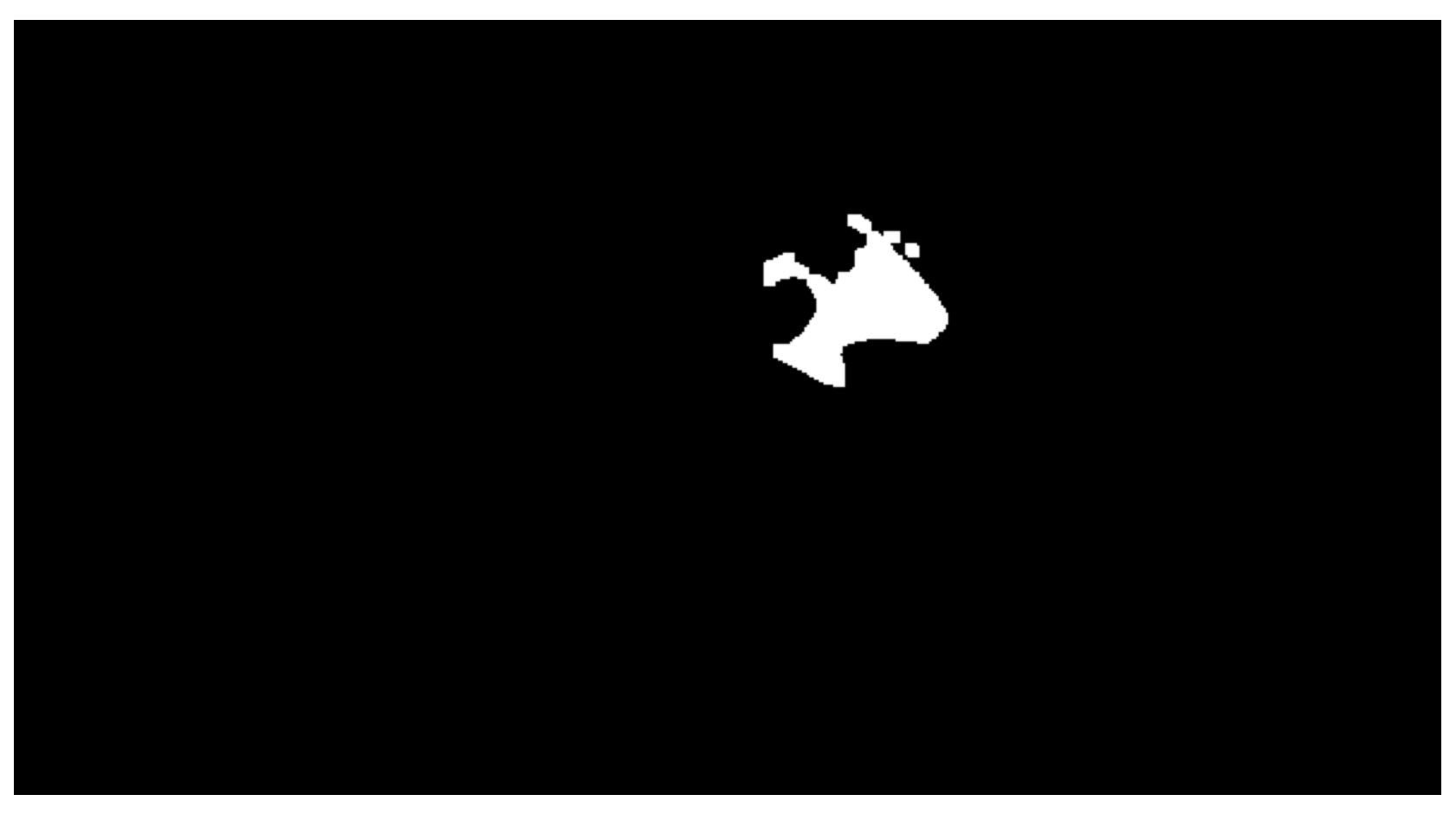
Introduction

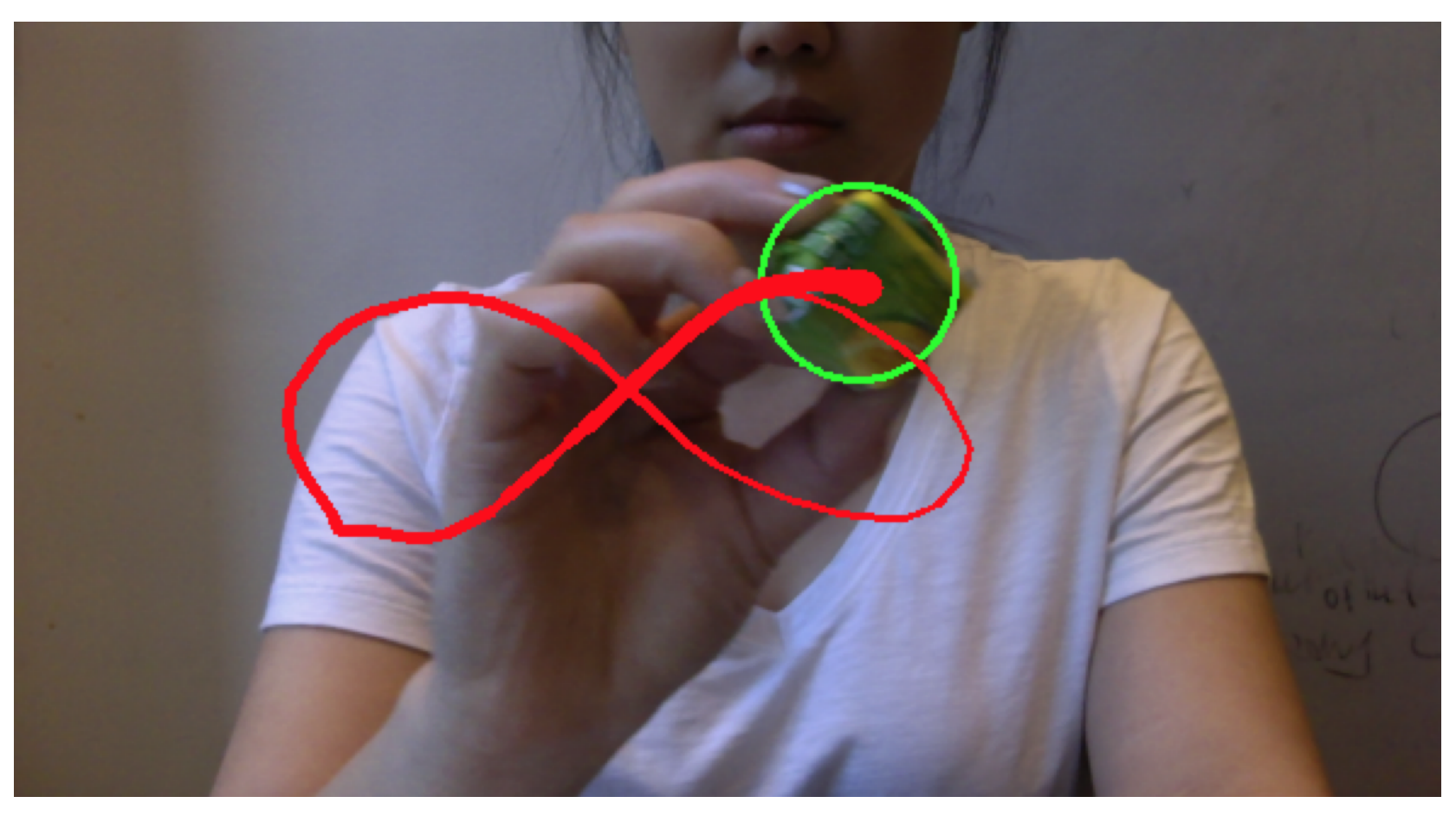
Our project aims to use a combination of computer vision and handwriting recognition to create a system that acts as a virtual whiteboard. Our model recognizes gestures written in the air and converts it into text. Users would be able to write "air words" facing a web-camera either real time or in-advance and have those gestures translated into character digits or words.

Air-writing recognition is an interesting topic, because it lends itself to many different future uses. Most intuitively, it allows for users with specific needs an alternative forms of communication. Virtual Reality systems such as the HTC Vive [1] provide a virtual whiteboard experience at a high cost that requires an extensive tracking system. Our system requires only a computer and camera, and therefore provides more affordability and accessibility.

This idea could also be extrapolated into using air gestures as a more universal input interface for technologies that are unsuited for the traditional keyboard and mouse.

Experiments





Result

Our air writing recognition program was able to successfully identify most character digits that we inputted. The algorithm classified lowercase letters the best, digits second best, and uppercase letters the worst. We speculate that this happens because of two factors: the number of strokes required to write the character digit and the amount of training data each class had.

We think that lowercase letters are identified most accurately because this class has the least number of strokes (most can be written in cursive-like fashion). The digit inputs did require clear segmentation between strokes, but the EMNIST dataset has far more digit samples than letter samples and thus– we speculate– the network was trained better for digits. The network struggled with uppercase letters. Letters needed to be written very carefully (and often slowly) as stroke placements are crucial in identifying capital letters.

Future Work

If we were to continue this project, we think that by better tuning the depth analysis, the results would be much better. To use our project (in its current state) the user must be careful and experienced in making their air-writing gestures. Since our goal is to introduce an input interface that allows for greater accessibility, the difficulty of use contradicts that.

To better match our project needs, we may choose to decrease the number of digit samples in our handwriting analysis model.

References

[1] *https://developer.vive.com/us/*

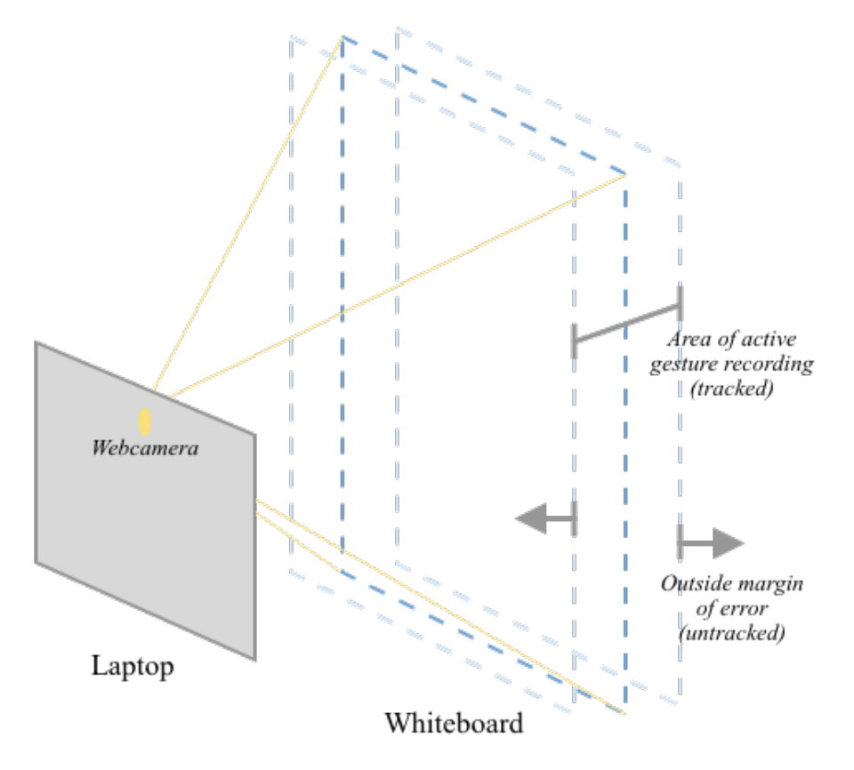
[2] *https://opencv.org*

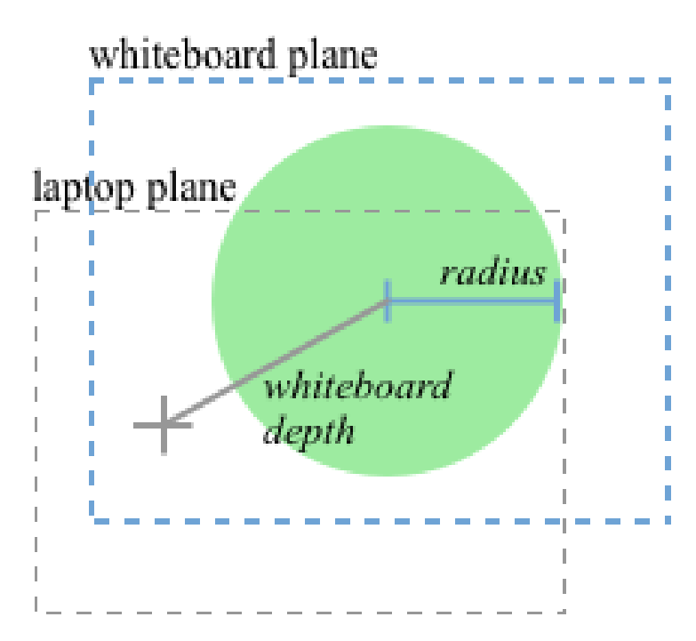
[3] EMNIST: an extension of MNIST to handwritten letters. *https://arxiv.org/pdf/1702.05373.pdf*

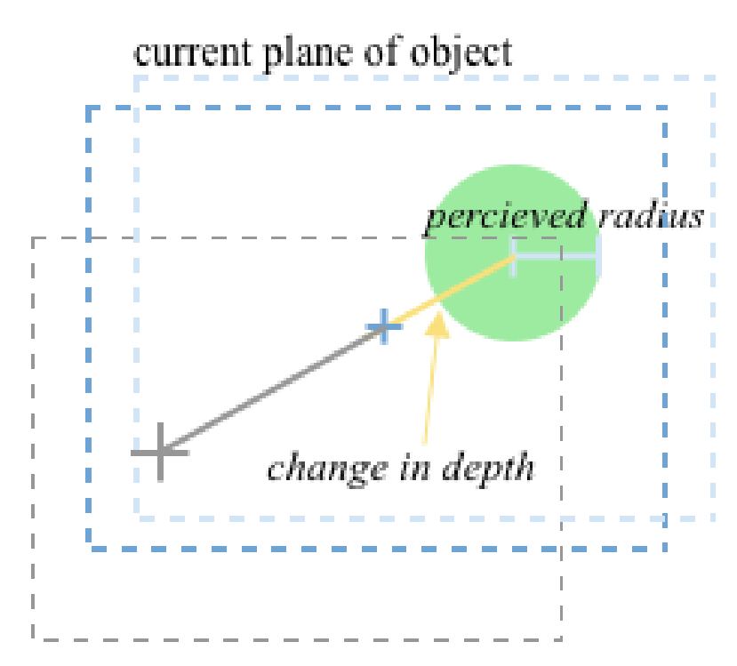
Motion Capturing

Tracking the object via the OpenCV library [2] through each video frame is done through preprocessing the frame image, masking the target object with a color range to detect it from the background, and finding the contours in the mask. The program continues to loop the tracking steps until it quits.

1. Set Parameter Values
   a. Define upper and lower boundaries for color of object.
2. Preprocessing
   a. Resize frame while maintaining aspect ratio.
   b. Apply Gaussian Blur with an 11 x 11 kernel to convolve the image and reduce Gaussian noise.
   c. Convert to HSV color space.
3. Construct Mask for Target Object
   a. Find object in range of preset color boundaries and create binary mask of object.
   b. Perform morphological transformations, erosion and dilation, to clean up mask.
4. Detect Contours
   a. Discover and select largest contour and compute minimum enclosing circle around object.
   b. Calculate and save the coordinates of the center of the object.
5. Classification Input Generation
   a. Join the saved coordinates and put them on a blackboard to pass into model.


Webcamera
Laptop
Whiteboard
Area of active gesture recording (tracked)
Outside margin of error (untracked)





Segmentation Problem
Since we are classifying uppercase letters and digits along with lowercase letters, we found we needed a method to separate different writing "strokes". Stroke segmentation can be thought of as instances where a marker would leave the whiteboard mid-character.

Depth Analysis
We settled on depth analysis to solve the segmentation problem because it is intuitive as well as an interesting technical problem. The classic stereoscopic approach involves at least two cameras and a complicated setup, which is relatively inaccessible and expensive. Instead, we utilized a simple mathematical property of proportions to calculate depth with only one camera. This allows the user to act as if they are actually writing on a whiteboard and then lifting the marker to start the next stroke– the intuitive action. Our solution works because our problem only involves calculating planar depth and assumes that the size of the tracked object remains constant.

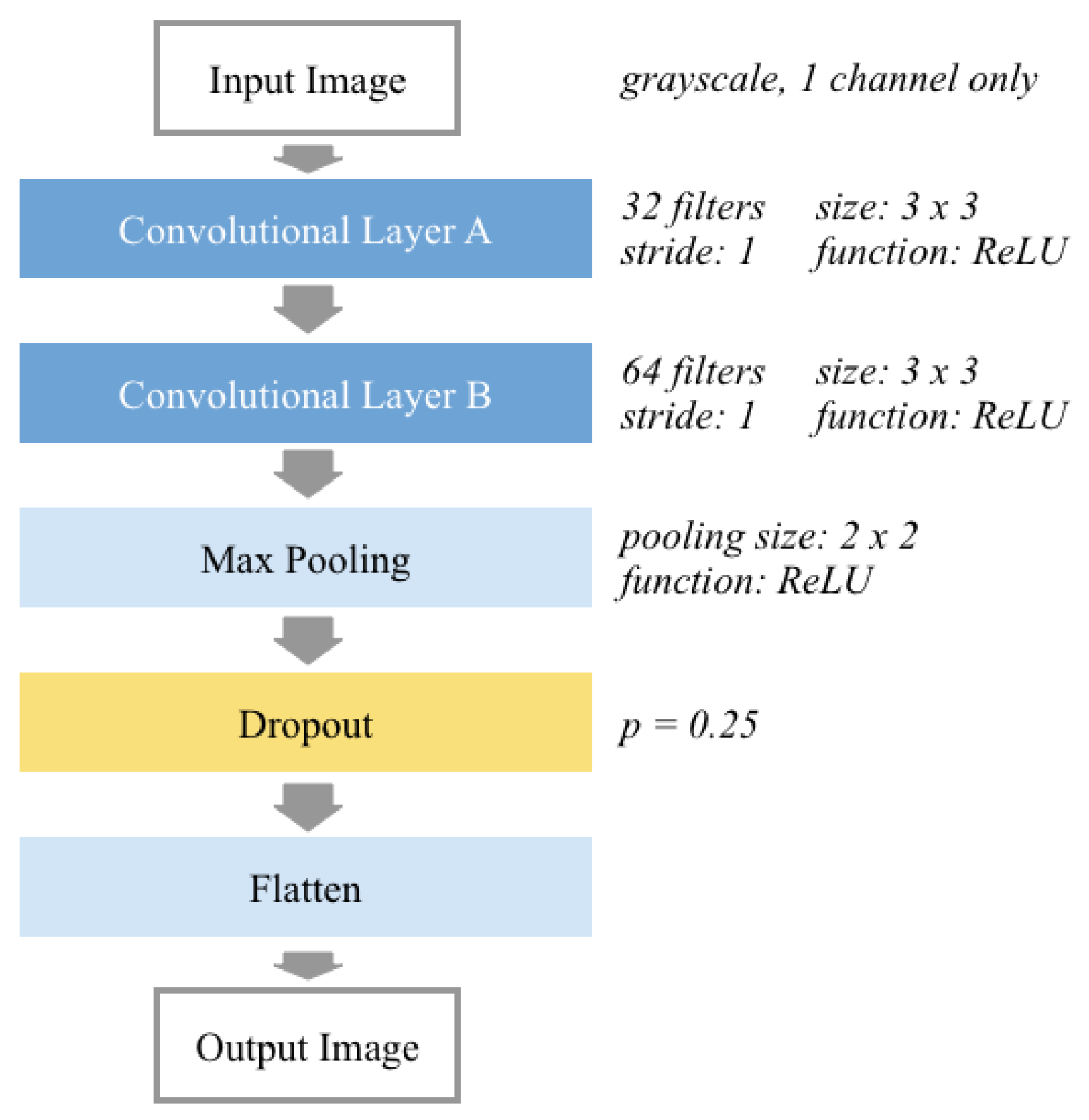
$$\frac{originalradius}{whiteboarddepth} = \frac{percievedradius}{whiteboarddepth + \delta depth}$$

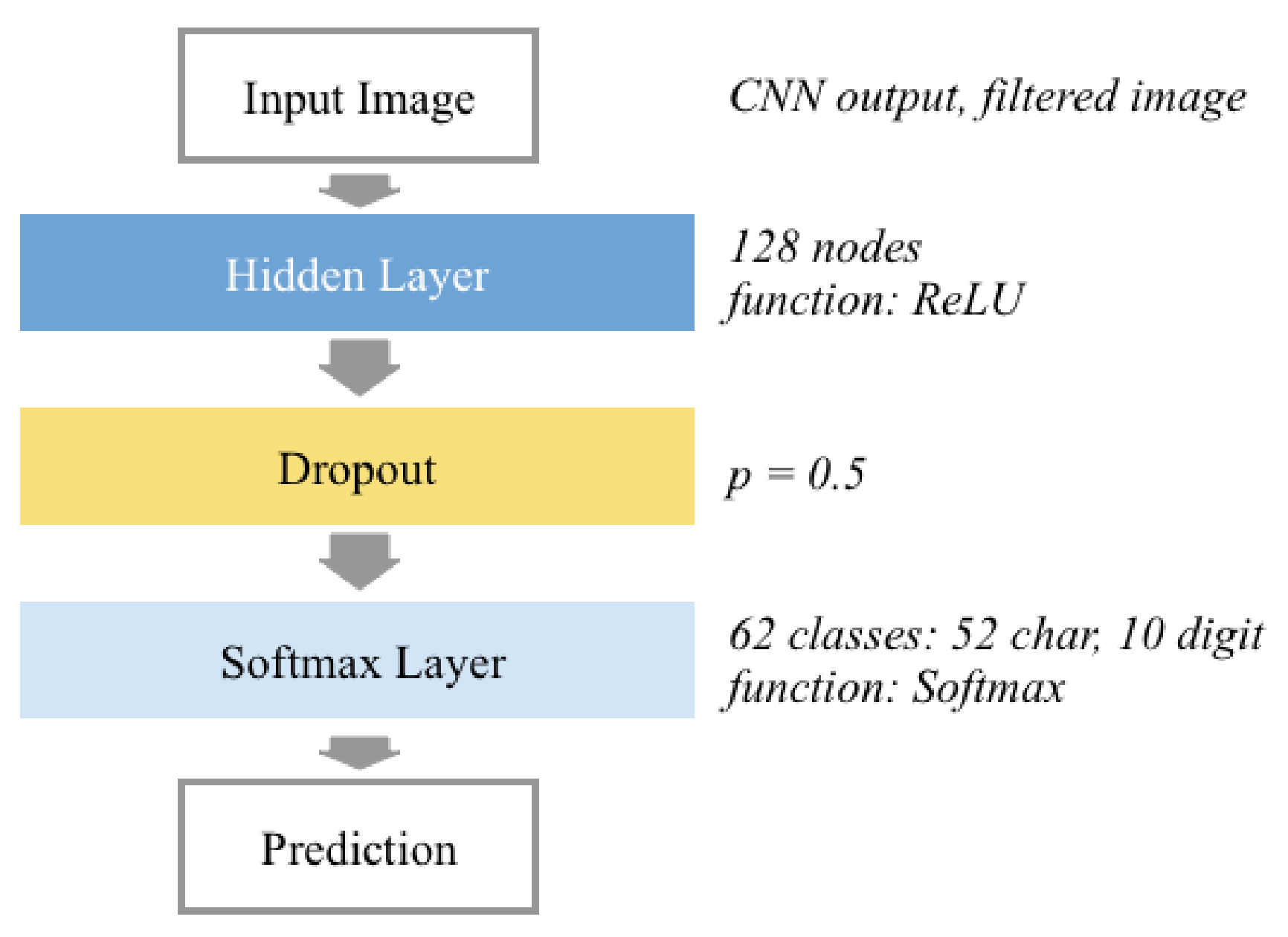
By solving for $\delta depth$ at each frame, we are able to track the changes in depth. In our program, we implemented a margin of error in which the gesture is tracked, and the circular bounding box changes color to notify the user if they exceed the margin of error.

Handwriting Classification

Dataset
The dataset we use for this project is the EMNIST (extended MNIST) Dataset of handwritten number digits represented in 28 by 28 pixel format [3]. This very popular dataset includes samples of the 10 single digits (0-9) and the 52 upper and lower case letters (a-z, A-Z). The EMNIST Dataset is split 9:1– 90% training and 10% testing. The EMNIST Dataset has far more digit samples than letter samples, and the ratio of each letter is approximately equal to the frequency of each letter in the English language.

Classification
Handwriting classification is a well-explored machine learning problem, and we knew that it was unlikely to create an architecture that would surpass current accuracies. Therefore, we chose to stick with a classic convolutional network with two convolutional layers, one max pooling layer, and a simple feedforward network. We also implemented dropout both before flattening in the CNN layer and before the softmax activation layer of the feedforward network. For hyperparameters, we chose not to focus too much of our time here (as training takes a good while). Thus, we continued to use the well-tested values shown in our diagram below.


Input Image
grayscale, 1 channel only
Convolutional Layer A
32 filters stride: 1 size: 3 x 3 function: ReLU
Convolutional Layer B
64 filters stride: 1 size: 3 x 3 function: ReLU
Max Pooling
pooling size: 2 x 2 function: ReLU
Dropout
p = 0.25
Flatten
Output Image


Input Image
CNN output, filtered image
Hidden Layer
128 nodes function: ReLU
Dropout
p = 0.5
Softmax Layer
62 classes: 52 char, 10 digit function: Softmax
Prediction