Braille Translator

Mino Nakura, Andressa Chan, Eliza Huang

CSE 455 Final Project

The Problem: Translating a given Braille image into English. Our motivation for this project stems from our interest in accessibility. In previous classes, capstones, and internships, we've always had to think about developing for a wide audience. Our Braille Translator is a way to raise awareness about the Braille language and gain more insight on the difficulties that visually-impaired individuals face. Throughout this process, we not only learned a lot about the Braille alphabet, but also about how to use openCV and its related functions to classify a given Braille character.

Technique: We split the process into many different steps: segmenting the braille text into individual letters, detecting and classifying each letter, and finally piecing each character together to formulate the sentence. To segment our images, we first figured out how wide and tall each Braille character is and broke up our images based on those measurements. We then passed in every image to our classifier function. In this function, we processed the image by dilating and eroding it. By dilating our image, we make the image features more prominent. Namely, we try to emphasize the black circles in our image. Since our image may contain unnecessary blank circles, we erode the image to diminish unnecessary noise. Then, we run Canny edge detection to outline the necessary circles that make up the letter. Finally, we find the contours on the image, which are the black circles. Using the coordinates of the found contours, we map the coordinates to the corresponding grid and encode the braille circles as a string of grid numbers. Then, we simply decode the encoded string to the alphabet character. We repeatedly do this for every letter and cumulate a string which represents the Braille translation.

Challenges: Since we wanted our translator to be dynamic and user-friendly, we decided to have a web app interface that makes it easy for anyone to use. However, one problem we immediately realized was that we had to find a framework that worked with openCV. After some research we decided on using Flask since its for Python and is known to be good for beginners since none of us have prior experience in Python web development.

For the braille detection, we considered two approaches to our problem: training a neural network or extracting features on each letter. We first attempted to train a neural network but were unable to find a dataset large enough to properly train it. Therefore, we tried to apply a Transfer Learning technique to resNET, but this also did not properly classify our images. This may be due to our lack of background in machine learning, which made it difficult to design and implement a convolutional neural network. This led us to attempt feature extraction as our main method of letter classification. One persistent issue we faced was falsely detecting circles in our image, which led to improper translations. We tried many different filters to fix this issue, including resizing, sharpening, and grayscaling. In the end, however, we found that Canny edge detection, coupled with erosion and dilation, was the most effective filtering method to properly detect the circles. We tried using the HoughCircles function to detect circles, but this proved to be difficult because there were many parameters we had to tune.

Finally, we also had issues piecing the frontend and backend together. We segmented our images and translated them into grayscale, but our translation method was only compatible with a colored image. We also struggled with file redirects and importing the correct classes because we ran the Flask app and the Python program in different directories.

Code Resources: In terms of the code, we wrote the majority of the backend. Our backend consisted of a Model.py class to segment the images into letters, a FeatureExtraction.py class to process and find the circles and encode a string, and a CodeConverter.py class to decode the string into a letter. Model.py and CodeConverter.py were written completely from scratch, and FeatureExtraction.py was mostly our own, but we didn't know how to use the findContours function from openCV, so we found some code online and based ours off that. For our frontend, we used bootstrap for the majority of the css but made custom css for some elements. In addition, the html and the rest of the .py frontend files and layout were based off the "Getting Started With Flask" scotch tutorial. Additionally, the file uploading code was done with the help of code from stack overflow.

The repository for our code can be found at https://gitlab.cs.washington.edu/nakuram/braille-iant.

References:

https://scotch.io/tutorials/getting-started-with-flask-a-python-microframework
http://www.cs.ubc.ca/projects/knoll/braille.html
https://gitlab.cs.washington.edu/nakuram/braille-iant
https://www.pharmabraille.com/pharmaceutical-braille/the-braille-alphabet/
https://stackoverflow.com/questions/19898283/folder-and-files-upload-with-flask