**Summary Report** combined with **Developmental Journal** as they are better understood together:

Objective: I really enjoyed both the edge detection homework and the text recognition homework we did. I thought it would generate better results to combine them.

Summary: My hunch was correct that it generated better overall results for text recognition than our naïve attempt previously. I used our alphabet and text images from HW4 and I modified the program to work with the canny edge detection. For final applications, I converted most code to functions so it can be used with any text images and their corresponding alphabets. I also added a user input section, lines 133-138, that will enable users to change the ground truth text or use the one we were using depending on their will.

My first attempt at this resulted in an oddly skewed accuracy for one image over the other. I modified the code for what created my templates in the following way:

1. I no longer needed to invert the image and generate a skeleton by thinning, simplified this process.
2. I needed to expand the bounding box to get the full edges of some letters, as a result the box is expanded by one. I had tried expanding by 2 or 3, but this decreased the accuracy despite it providing the full edge tracing.
3. Each template for the alphabet is created by the canny edge detect function I created in the other HW. The threshold is now an optional variable, but with the current setup, 1 is the optimal threshold value. Though in earlier iterations, depending on our comparison tools, either the shape or the amount of non-zero values, in the bestmatch function we had to either drop the threshold to .5 or .75, which resulted in higher accuracy for either the alphabet or the textimg.
4. I also attempted to fill the template, I browsed the Scikit documents to see if there was anything that could, ultimately finding floodfill, although I elected not to utilize this as I did not see a need.
5. For the bestmatch function, I was having issues with line 214, which produced the error of axis out of bounds. This was solved by inverting the image on line 210.
6. I ultimately decided to use cv.countNonZero() to compare which template best matched, as the shape of the template can lead to templates with similar shapes overriding each other. With nonzero values this ended up being less likely in the current version, though there are some instances.
7. My biggest issue was the fact that I was obtaining an accuracy between 70-87% for the text image and an accuracy of -8-40% or vice versa when I would modify the bounding boxes or thresholds, for labelbyrows, or for canny. This issue was resolved by changing the thresholds on lines 144 and 151. I was then able to obtain accuracy of ~92.31 for the alphabet, and ~84.63 for the text image.
8. Scale also played a large role in this. If I didn’t scale my template accuracy was very low, but when I scaled by 3 or 4 I archived better results as then the edges were more clearly defined or actually existed.
9. I also adjusted the sigma in the canny function, though .4, which was what I preferred in the original project for it, worked best. It can be modified as needed now as an optional parameter.

Once adjusting for point 7, I added more user-friendly pieces to make the code more accessible for other images. I maintained the accuracy level through these adjustments.

1. I added line 138 in order to filter out the punctuation within the ground truth text. The string python lib has a punctuation filter which is what I used.
2. I converted blocks of code into functions to easily allow for different images to be used.
3. I allowed for user input to change the ground truth. The file names could also be input as user input and then this program could run without having to manually change that but I elected not to change that at this moment.

I am still having issues or could improve in the following places:

1. The alphabet visualize shows me that it does not recognize 4 letters. This seems odd to me when you examine the text image, specifically for the letter ‘a’ which doesn’t have a red match in the alphabet, but it does for the text image. The ‘g’ matches in the alphabet image, but not in the first section of the text image in the word “Pangram”, “G” doesn’t match in the alphabet image, but it matches in the text image. I have attempted to solve this issue but am unsure why it is occurring. I speculate it has to do with the text size, or even the bold nature of the text, but despite modifications made to this, I am still unable to get those items to match.
2. The statistics. I have attempted to rework what I had. But overall, it did not work well for me and would prefer to use the templates.
3. I attempted to get better matches with just scaling. As Emi and I explained in office hours, we looked through my code but it for some reason tanks my accuracy when I just scale it. final2.py is the code I used to test this, and I am curious as to why it isn’t working. I spent a long time changing scalers, thresholds, even line position. Unfortunately, it still does not work correctly, so I have not seen a 100% accuracy level yet.
4. I did look through chamfer distance, but mostly just to do the reading on it. Scipy had info on the distance transform which I have linked below in my sources.
5. I would like to experiment, when I have more time, with the method we talked about in office hours where you extract vertical image slices for the letter profile and then compare the cumulative sum of the pixels. That method seems really interesting, and I am curious as to how that accuracy level looks.
6. I did also utilize my “extra” best match function from HW4 to see exactly what letters were matching with what and that was helpful, and probably could assist in fixing issue 1 above, but I would have to make several modifications I am unsure of.

Reflection:

Overall, I really enjoy text recognition and I feel like I have managed to implement it successfully here different from our original. It did take a lot of little tweaks which is definitely the hardest thing for me just because I find most of my mistakes live in the details of my code I hardly can see after looking at it for so long. So despite having the functions laid out from the homework I still had to make a lot of very small changes with very large impacts. I honestly really like working with this and if there’s any opportunity for me to build upon it I would really love to pursue that. I will say that there are some kinks that need sorting out but this is definitely a feasible application.

Sources:

1. For the countNonZero function I found it on stackoverflow here: <https://stackoverflow.com/questions/28981238/how-to-count-the-number-of-non-zero-pixels-of-the-canny-image-in-my-python-progr>
2. Distance transform CDT: <https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.distance_transform_cdt.html>
3. Looking through region props stats these definitions were helpful along with the scikit definitions: <http://www.ece.northwestern.edu/local-apps/matlabhelp/toolbox/images/regionprops.html> and <https://scikit-image.org/docs/stable/api/skimage.measure.html#skimage.measure.regionprops>