PROJECT PLAN AND PRELIMINARY EXPERIMENTS

Team: reCAPTCHA
Members: Johann Ryan, Rithvik Subramanya, Indresh Srivastava, Dominic Grazioli

Project Plan

Week 8:

- Separation of individual characters (completed) - 3 hours
- Features extracted and neural network setup - 3 hours

Week 9:

- Finish neural net training - 4 hours
- Test our neural net on individual characters - 3 hours

Week 10:

- Stitch individual character identification to identify all text in the captcha image - 3 hours
- Prepare for the presentation - 2 hours

Preliminary Experiments

We found a couple of kaggle datasets (hard - https://www.kaggle.com/fournierp/captcha-version-2-images, easy - https://www.kaggle.com/genesis16/captcha-4-letter) with captcha text images labeled with their text values. We started our experiments by first pulling in all these images into MATLAB. While doing so we found that our images were not all of one single type (jpg and png), so we wrote a script to handle the creation of a single set of all images, of all types, to hold grayscale image values for every image. While doing so we also created a set holding labels for all these images for training and testing (we plan to use this set as a map for our image set). The following code handles the creation of the two sets we spoke about earlier:

```
clear all;
if ~isfile('captcha_data.mat')
    % Import all images with their labels
    path = "reCAPTCHA/images/";
    files_png = dir(fullfile(path,"*.png"));
    num_of_png_imgs = numel(files_png);
    files_jpg = dir(fullfile(path,"*.jpg"));
    num_of_jpg_imgs = numel(files_jpg);
    images_png = {};
    labels_png = {};
    images_jpg = {};
    labels_jpg = {};
    for i = 1:num_of_png_imgs
        filepath = fullfile(path, files_png(i).name);
        str = strsplit(files_png(i).name, '.');
        labels_png = [labels_png, str{1}];
        img = rgb2gray(imread(char(filepath)));
        images_png = [images_png, img];
    end
    for i = 1:num_of_jpg_imgs
        filepath = fullfile(path, files_jpg(i).name);
        str = strsplit(files_jpg(i).name, '.');
        labels_jpg = [labels_jpg, str{1}];
        img = rgb2gray(imread(char(filepath)));
        images_jpg = [images_jpg, img];
    end
    images = [images_png,images_jpg];
    labels = [labels_png, labels_jpg];

    % Export grey scale images and their labels to file
    save('captcha_data.mat','images', 'labels')
else
    load('captcha_data.mat')
end
```

Figure 1: Importing images

We then thought it would be easier to analyze images for these texts if we break the image up into segments containing single characters. But before that, we need to run edge detection to see if we can reduce the noise present in the image. Thus we used simple canny edge detection on our images for edge detection.

Figure 2: Before and After Canny Edge detection (EASY)

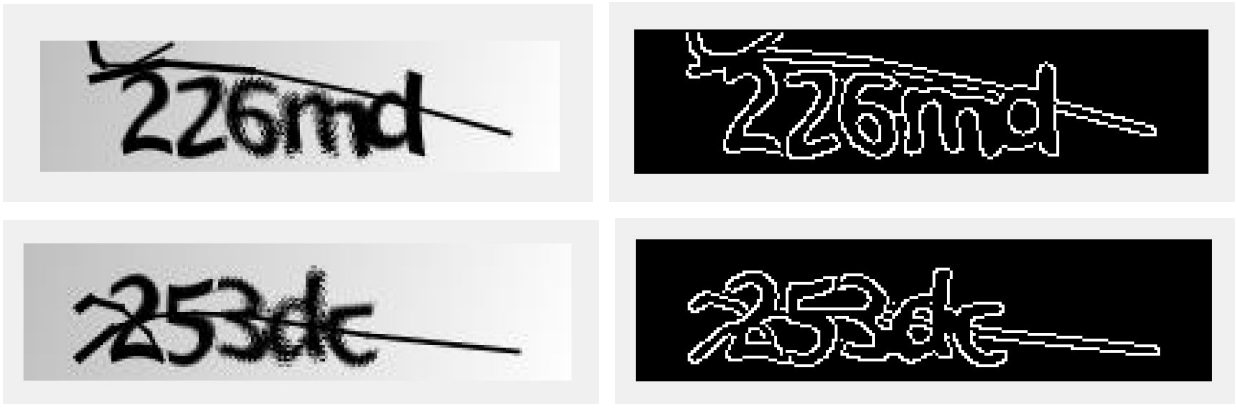
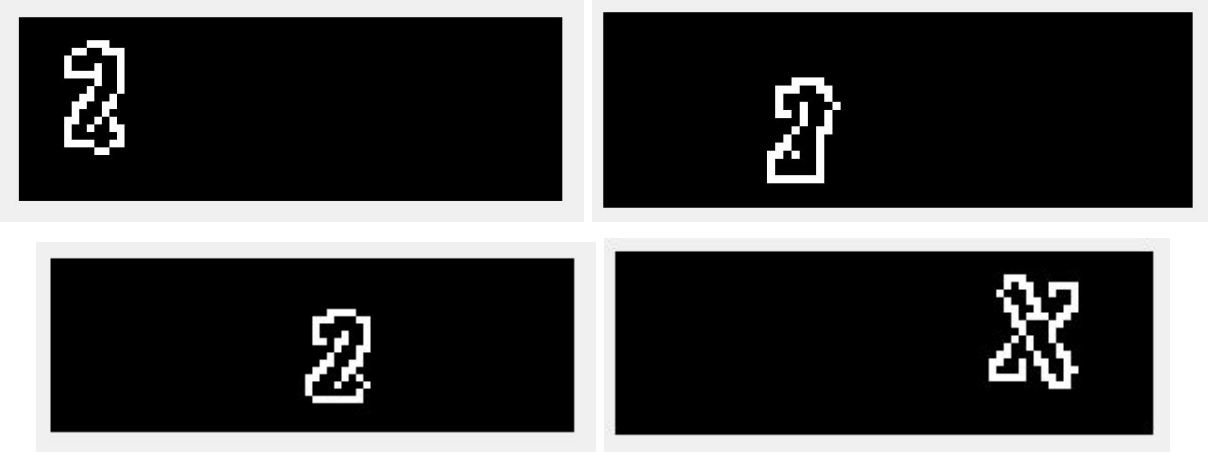


Figure 3: Before and After Canny Edge detection (HARD)

Here we can see that it is easier to read the images once canny edge detection has been run on them. So we hope that our segmentation on these images after edge detection should be able to more accurately identify the characters.

For segmentation on the easy set of images we use bwlabel to isolate each label we encounter. We now are planning to generate a set of these single characters with their labels and feed it into a neural network. Once we have our network trained to identify these individual characters we can then use that information to stitch together the textual contents of the entire captcha image. The following are results from bwlabel:

Original Image | After Edge Detection



Bwlabel to isolate characters

Figure 4: Process of isolating characters

Figure 5: Boundary box on characters

Although we were able to isolate the letters using bwlabel, we also wanted to accommodate for the empty space in the images. Our intention behind this is that having letters floating around in the image at different points in space would make it harder for an SVM or CNN to pull features from (although a CNN might be able to figure it out). Hence, we felt the best approach would be to establish a bounding box around the bwlabel separated letters using the regionprops function. With the coordinates of the corners of the bounding box, we can use the imcrop command to crop the images to just the letter. Once the letters are cropped, we can scale them to the same size using imresize command or by adding black space to set the size to that of the

largest image. Ultimately, both processes will have the letter centered which will help the SVM/CNN during training, although the scaling might be trickier since it does alter the letter itself.

Next, we automate this process to create a list of letter images and use a similar process on the filenames to get the corresponding label for each letter. With this, we can enter this image data into a CNN to create a basic letter classifier (which should be pretty easy for the standard letter fonts used in the easier data set). With a properly trained CNN, which uses our process above to split apart any new text image and plug each letter into a CNN to obtains the corresponding text. The one potential issue with this approach over using the CNN on the text image as a whole is that the error of the CNN would be raised to a power of how many letters are in the text, but we strongly feel that the sheer accuracy improvements with a single character classifier would be enough to mitigate this effect (this is also strongly supported in the papers we read). Furthermore, using tricks like flipping or rotating certain symmetric letters in the training data can improve the training process and thereby the accuracy of the model.

Clearly, the challenge here is adopting a similar process for the harder dataset, where the letters aren't clearly separable. One of our initial thoughts for doing this would be to first specify the region where the text exists and then divide that region by the number of characters to get approximate letter images. This would work reasonably well since from what we observed, the letters are all approximately the same size and a well-trained CNN shouldn't have too much difficulty with some parts of the letters are cut off. However, one difficulty with this is that for most images in the hard dataset, there is a diagonal line that runs through the text that makes it difficult to programmatically crop just the text out. This will be something we must further consider throughout the project.