CS771 Assignment 3, Group 21

Abhyuday Pandey	Ayush Kumar	Deepesh Lall	Srinjay Kumar	Umang Malik
170039	170195	170236	170722	170765

1 Breaking the Captcha

The steps required in the breaking of captcha are as follows :

- Image Preprocessing
- Character Prediction

2 Image Preprocessing :

We use the opencv library for all the steps involving image processing.

We will depict the steps of image processing on the Captcha ABJG.png in train.

The steps involved in the breaking of the captcha are :

- Turning the image to grayscale
- Dilating the image
- Floodfill
- Edge detection
- Detecting Contours
- Changing background colors
- Segmentation of image

The description of all the steps are as follows :

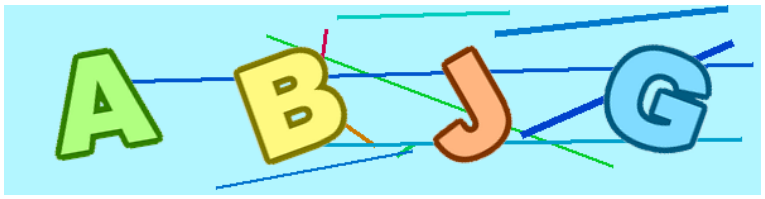

Figure 1: An image of Captcha

Grayscaling
Grayscaling is used in our case for converting from RGB color code to the the shades of gray[1]. The image colors would now vary from white to black. The color code change is also required because the canny edge detection algorithm requires grayscale images for edge detection to take place.

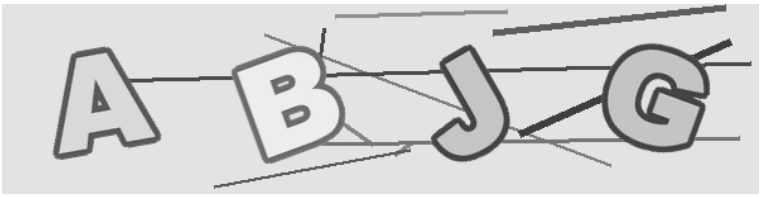

Figure 2: An image of GrayScaled Captcha

The image after grayscaling is depicted in Figure 2.

Dilation
This procedure involves convolution with a kernel of specific size. In this case, the pixels in the picture are replaced with anchor at the center[2].
Hyperparameter tuning: We used a MORPH_RECT rectangular structure with dilation size 1 and kernel size 3×3 with centre as the anchor. We used a trackbar and observed that this hyperparameter worked the best.

Hyperparameter	Optimum Value
dilation size	1
kernel size	3×3

The image after dilation is depicted in Figure 3.

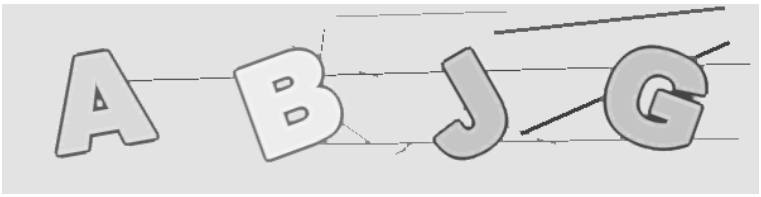

Figure 3: An image of dilated Captcha

Floodfill
This feature starts from a pixel in one of the corners and follows a graph traversal algorithm and changes the pixel value of all the pixels that are same as that of the corner pixel. In this case, we make all of them white. This can be visualized as the typical floodfill used in MS Paint.
Hyperparameter tuning: We used 255 as the background colour.

Hyperparameter	Optimum Value
BGColour	255

The image after implementing floodfill is depicted in figure 4.

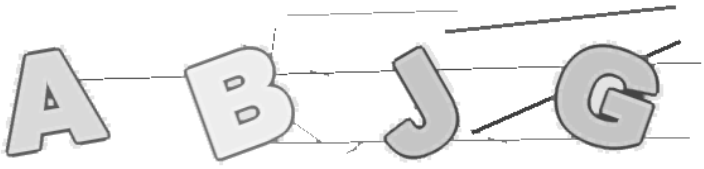

Figure 4: An image of Floodfilled Captcha

Edge Detection

For edge detection, we use the canny edge detection algorithm[3].
The steps in the canny edge detection algorithm are :

- Noise Reduction.
- Finding Intensity Gradient of the Image.
- Non-maximum suppression
- Hysteresis Detection

Hyperparameter tuning: The threshold 1 and threshold 2 for hysterisis is varied and (30,200) was found to be the optimal.

Hyperparameter	Optimum Value
Threshold 1	30
Threshold 2	200

The image after canny edge detection is depicted in image Figure 5.

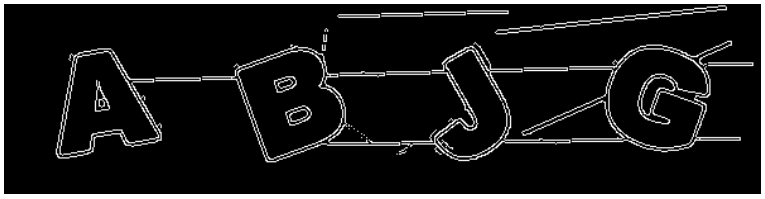

Figure 5: An image of Canny edge detected Captcha

Detecting Contours
Next, we use the detected edges to find the contours in the image. We use two functions for contour detection and drawing:

- findContours
- drawContours

The image after detecting contours is depicted in Figure 6.

Figure 6: An image of Contours detected Captcha

Changing Background Colors
We change the background color of the image to white. All the pixels that have values equal to that of the extreme top left pixel are considered to be belonging to background.
The image after changing background is depicted in Figure 7.

Segmentation of Images
The image is segmented with the help of detection of continuous streaks of non-white pixels. According to our algorithm, if the continuous streak of non-white pixels exceeds 9, then we designate it to

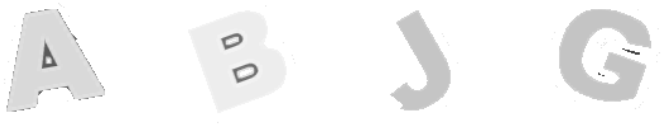

Figure 7: An image of Captcha background color changed

be an character.
Split is found using modified horizontal-histogram. The segmented characters we get are similar to what we have depicted in Figure 8. Each image is then resized from 150×150 pixels to 30×30 pixels.

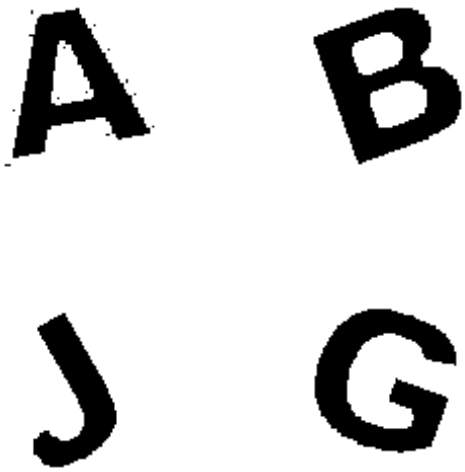

Figure 8: Image of Segmented Captcha

3 Character Prediction

We make use of **Neural Networks** for the prediction of characters.
We used libraries keras[4] and tensorflow[5] for our neural networks.
There are 3 layers in our neural networks :

- **Input Layer** : It contains 30×30 nodes corresponding to image pixels.
- **Hidden Layers** : It contains three hidden layers. The first hidden layer is Conv2D layer which applies a convolution of kernel size 5×5 on the input layer. The next hidden layer performs a max pooling on the previous hiddden layer. The MaxPooling2D layer is followed by a Flatten layer which converts the 2D matrix into a 1D feature vector.
- **Output Layer** : It contains 26 nodes corresponding to each character. It is densely connected to its previous layer.

The characteristics are as follows :

- There are 5 layers, namely, Input layer, Conv2D layer, MaxPooling2D layer, Flatten layer and Output layer.

- Output nodes give a probability distribution over all the characters.
- The character corresponding to the maximum probability is the predicted character.

Hyperparameter tuning : We tried with 1 hidden dense layer having 900 neurons. The model size as well as time was overwhelming. So, we removed dense perceptron layer and replaced it with sparse convolution layers with kernel size 5×5. The number of $epochs$ were varied and 40 was found to be the optimal value. The tuning for activation is, as follows,

Activation at Hidden layer 1	Activation at Output layer	Accuracy(%)
$reLU$	$softmax$	99.85
$softmax$	$reLU$	95.55
$softmax$	$softmax$	94.2
$reLU$	$reLU$	98.45

We used Categorical Cross Entropy as loss function, and Adam's gradient descent for performing optimization.

Hyperparameter	Optimum Value
Input layer size	30×30
Number of hidden layers	3
Activation at Hidden layer 1 (Conv2D)	$reLU$
Activation at Output layer	$softmax$
Loss function	Categorical Cross Entropy
Descent	Adam's gradient descent
epochs	40

References

References to papers and applications we used for studying and developing our prediction code.

[1] [GrayScaling]

[2] [Dilation]

[3] [Canny Edge Detection]

[4] [Tensorflow]

[5] [Keras]