# CS 771-A Assignment 3 Submission

**Team Yo**
Sarthak Kohli (200886)
Dishay Mehta (200341)
Shubhan Ravi (200971)
Mohit Gupta (200597)
Shrey Mehta (200580)

## Abstract

Report for the third assignment of Introduction to Machine Learning (CS771A) 2022-23-I offering.

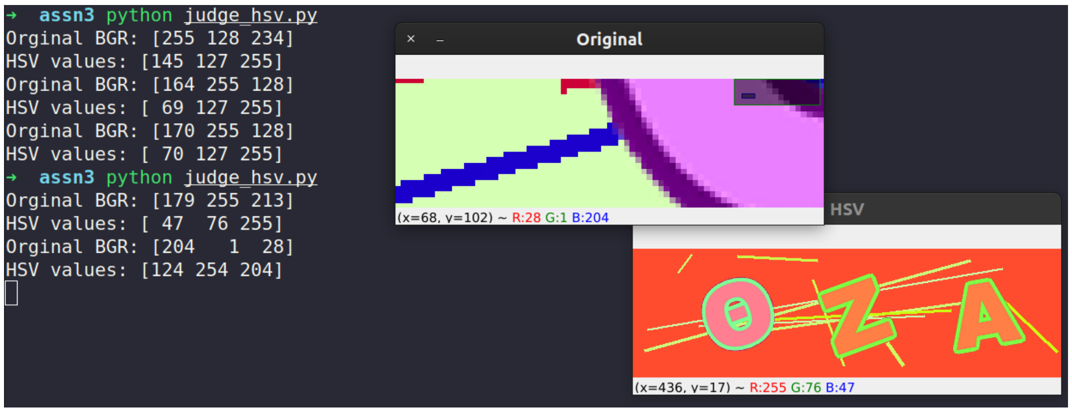
## Question 1

Describe the method you used to find out what characters are present in the image. Give all details such as algorithm used including hyperparameter search procedures, validation procedures.

### Solution

### Pre-processing

We observed on the train set that, on converting all the images to HSV color-scale, all the characters of all the images had the same (Saturation, Value) values of (127, 255) but different H values, which is different from background and stray lines. The first 3 HSV values in the image below are from each of the characters and the next 2 are from the background and a stray line.



```
➜  assn3 python judge_hsv.py
Orginal BGR: [255 128 234]
HSV values: [145 127 255]
Orginal BGR: [164 255 128]
HSV values: [ 69 127 255]
Orginal BGR: [170 255 128]
HSV values: [ 70 127 255]
➜  assn3 python judge_hsv.py
Orginal BGR: [179 255 213]
HSV values: [ 47  76 255]
Orginal BGR: [204   1  28]
HSV values: [124 254 204]
```

Original
(x=68, y=102) ~ R:28 G:1 B:204
HSV
(x=436, y=17) ~ R:255 G:76 B:47

Figure 1: HSV values of various elements

**Isolating Characters:** So, we take a training image **I**, make a corresponding mask (say **M**) of all the pixels with that (Saturation, Value) values of (127, 255). This mask **M** will only be used from now and contains just the characters.

Now, we have an image containing only characters without background and stray lines.

**Islanding:** We now take a null matrix (say **B**) of size 500 x 150. **B** will be used to mark indices visited during DFS in next step. Now, we go though every coloumn of **M** column by column, and if we hit a *non zero unvisited (B stores visited indices)* value, we start a DFS and mark all the indices it reaches by a unique number in **B** (100 for the first character, 150 for the second, 200 for the third). If we see that an unvisited non zero index lies between visited pixels( for example the middle line in THETA), we do DFS and mark it with the same number it lies in between in that coloumn (100 for the first character, 150 for the second, 200 for the third). Once we see that a coloumn is totally 0, the number with which the new indices of the new character will be marked is also changed.

**Segregation:** **B** now contains all the characters of **I** where, the first character is made of 100s, the second character is made of 150s, and the third character is made up of 200s. This gives each character a unique value. Now we produce masks of each character using OpenCV for the values 100, 150, 200 separately. We get 3 masks, say **M1, M2, M3**, each containing only one character and the space of other characters is blank.

**Cropping:** Cropping each of **M1, M2, M3** with a margin of 10px using the left-most, right-most, bottom-most and top-most *non zero pixel*. Now we get new matrices **C1, C2, C3** each containing only one letter. Now, we make each non zero pixel of each of **C1, C2, C3** as 1. So, basically till now, we have broken down image I into matrices **C1, C2, C3**, where **C1, C2, C3** contain the cropped first, second and third character of I respectively and is made of 1s. We now resize each image **C1, C2, C3** into square matrices of size 13x13 using OpenCV. Initialy 32x32 was chosen to be the size of character images but finally settled on 13x13 as it gave equivalent accuracy on the validation set and we needed to reduce space of our submission.

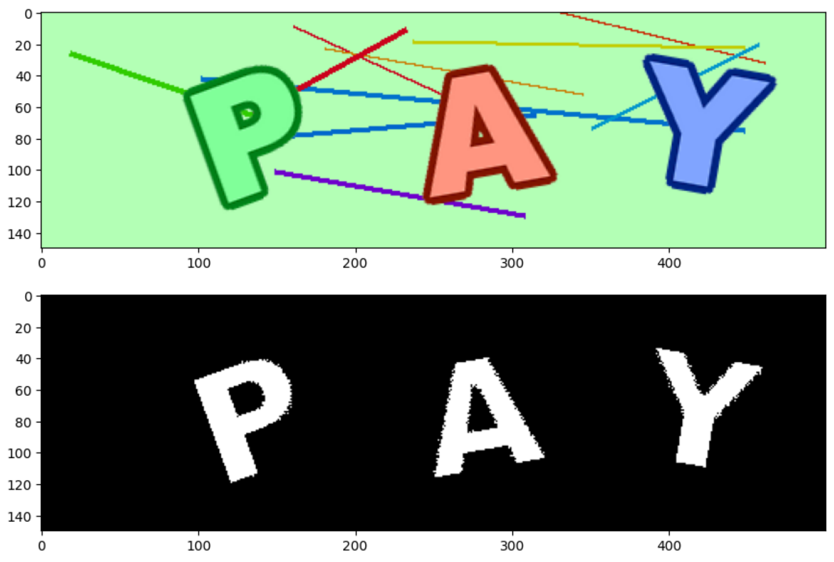


Figure 2: Original Image **I** (above) and **M** (below) of size 500x150

## Training our CNN

The new **C1, C2, C3** will be used to train the CNN to identify each character as we have the labels also of each of them from *labels.txt*. We do the pre-processing steps for every train image we use and we separated all the train images into their individual characters. We chose a CNN containing one input layer of 13x13 nodes, followed by a convolutional layer of 6x6 nodes that was followed by flattening, and one output layer of 24 nodes. The CNN modelled was created through *tensorflow.keras.layers*. There are no activation layers used. For a character, the node of output layer giving maximum value is chosen as the prediction. This CNN was trained for 13 epochs. For each epoch, a random 80% of all the characters (4800) was used for training and the rest 20% (1200) was used for validation. We saw that after 13 epochs, the CNN reached 100 percent accuracy on both training and validation images.

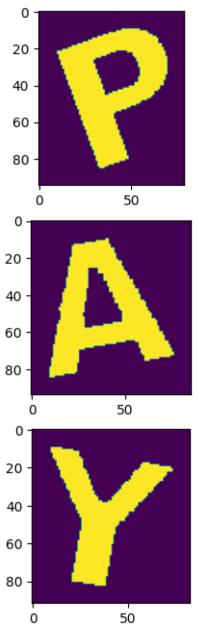


Figure 3: Intermediate cropped **C1, C2, C3**

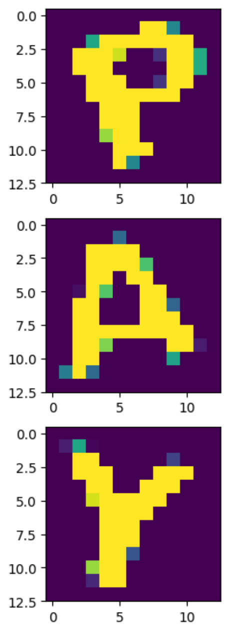


Figure 4: Scaled and cropped **C1, C2, C3**

The model initially had an input layer of 1024 (32x32) nodes, a two convolutional layers of 32 and 16 filters, followed by a dense output layer of 24 nodes. This model had a size of 840kB. So, we **scrapped** the hidden layer and tried again. This reduced the size further. This was followed by making the input size to 13x13 and this reduced the size even further while maintaining accuracy. Finally we obtained a model sized 41kB. We tried many input later sizes like 19x19, 17x17 and finally settled on 13x13. We could have made the model smaller but we chose to stop.

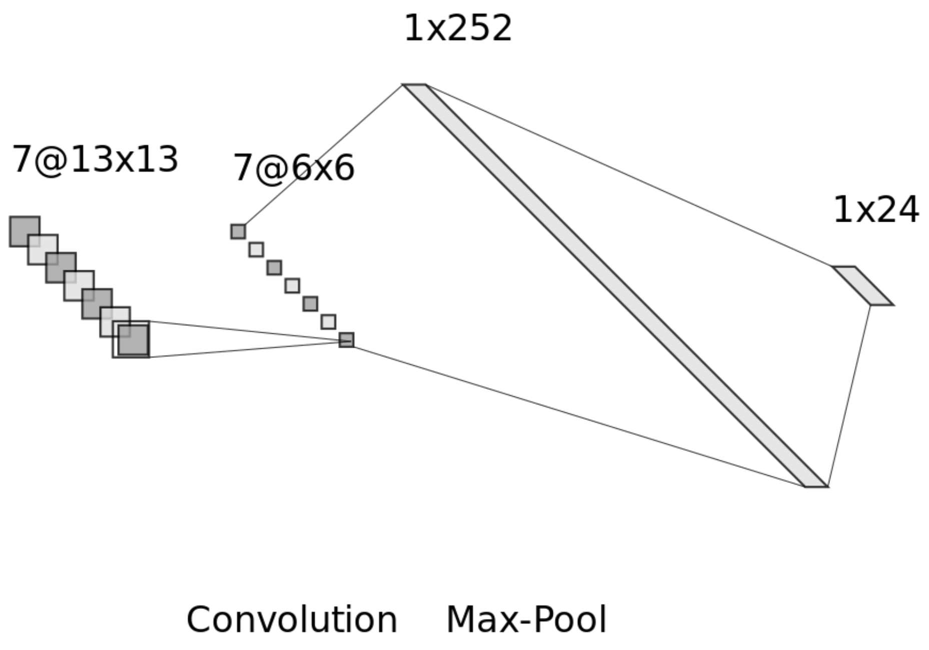


Figure 5: CNN Model Architecture

**Running test images using eval.py**

Now for each test image inputted from *eval.py*, we do the same pre-processing steps as above and find out the corresponding **C1, C2, C3** matrices and run it through the model and return the corresponding labels of each character.

Assuming that the (Saturation, Value) of HSV values of all the characters of all the test images are same (same property as the given train images), we extract the S,V of the characters of the first test image encountered. We do this by sorting the counts of all unique pixel HSV values of the image. The first one would be the background color. And the second rank must be the filled color of the one of the characters. So, we set this to be S, V for all characters of all images from now for the pre-processing.

## Credits

Our friend B.Anshuman who already had some experience with Tensorflow and OpenCV actively guided us throughout this assignment.