COL 783

Assignment 4

Rajbir Malik
2017CS10416

November 3, 2019

Robust Form Processing

Overview

The assignment required us to process some forms in different natural settings, and trying to extract meaningful information from the forms.
Being much open ended, we had the opportunity to try various procedures to get the task done. I was, after many attempts, able to come up with a "mostly" automated process to get the required information.
I have presented this approach and explained the procedure in the upcoming report. I've also discussed another approach that was reasonable to some scale, but couldn't be scaled, by me, to the desired level.

Image Alignment

I worked upon the lines of a standard form scanner, to get all the input images aligned correctly. Having created a template-default form, I used it as reference to get the ideal warp-perspective for the input image, with the help of descriptor-matching.
The exact pseudo-code for the same has been described below...

```
def image_alignment(_input_image, _template):
    - Create ORB Descriptor
    - Get Key-Points and Descriptors
    - Match the features
    - Filter the matches, to get the ideal ones for mapping
    - Using keypoints, discover ideal homography
    - Use the homography, and warp the input image
    - Return the warped image
```

Results

The results for alignment procedure were quite-impressive, achieveing as good as 99% accuracy for `printouts` and `scanned` and reasonable results for `boooklets`. Perhaps, if I could find ideal templates for booklets as well (which I couldn't), then there too a decent result may have been procured.
Further, I present the results of alignment on all the categories (2 of each kind).

Another Approach

I also tried with a much simpler approach earlier, which involved finding the lines using `Canny-Edge Detector` and then using `Hough-Transform` to get the angle at which maximum no. of lines appeared.
This approached, though naive seemed to work sensibly for `scanned`-images, but failed badly for both `printouts` and `boooklets`. So, I was compelled to drop the idea.

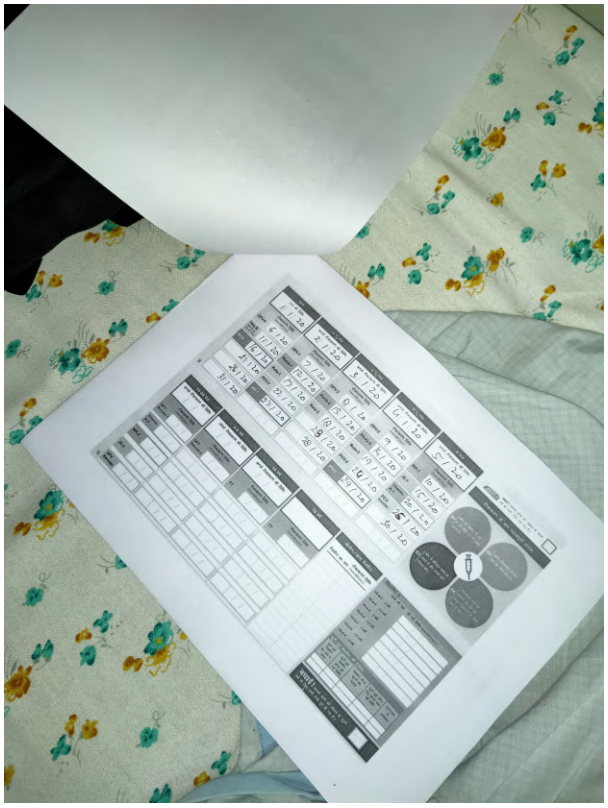

Figure 1: Printouts$_1$

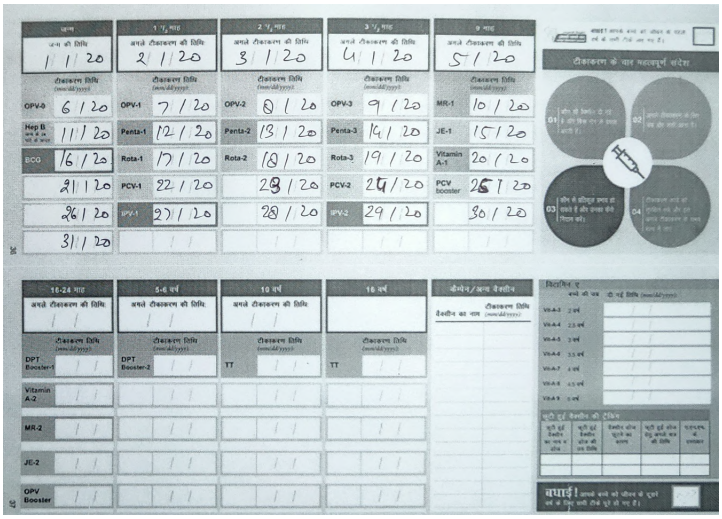

Figure 2: Aligned

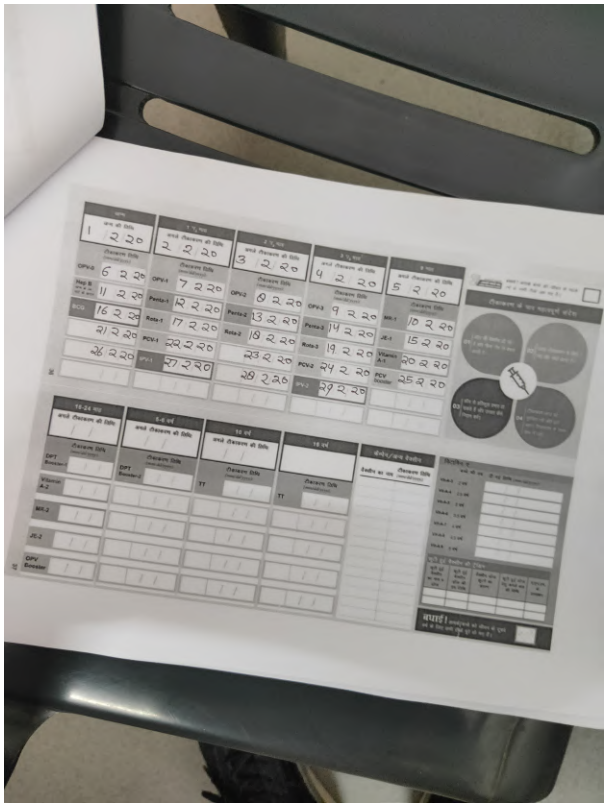

Figure 3: Printouts$_2$

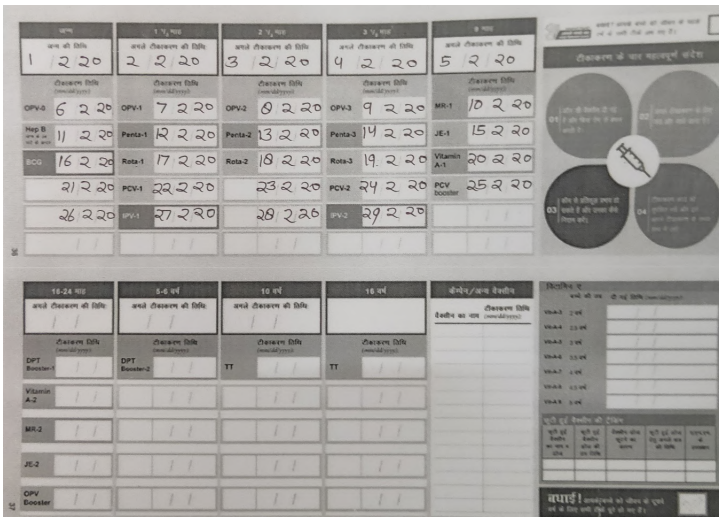

Figure 4: Aligned

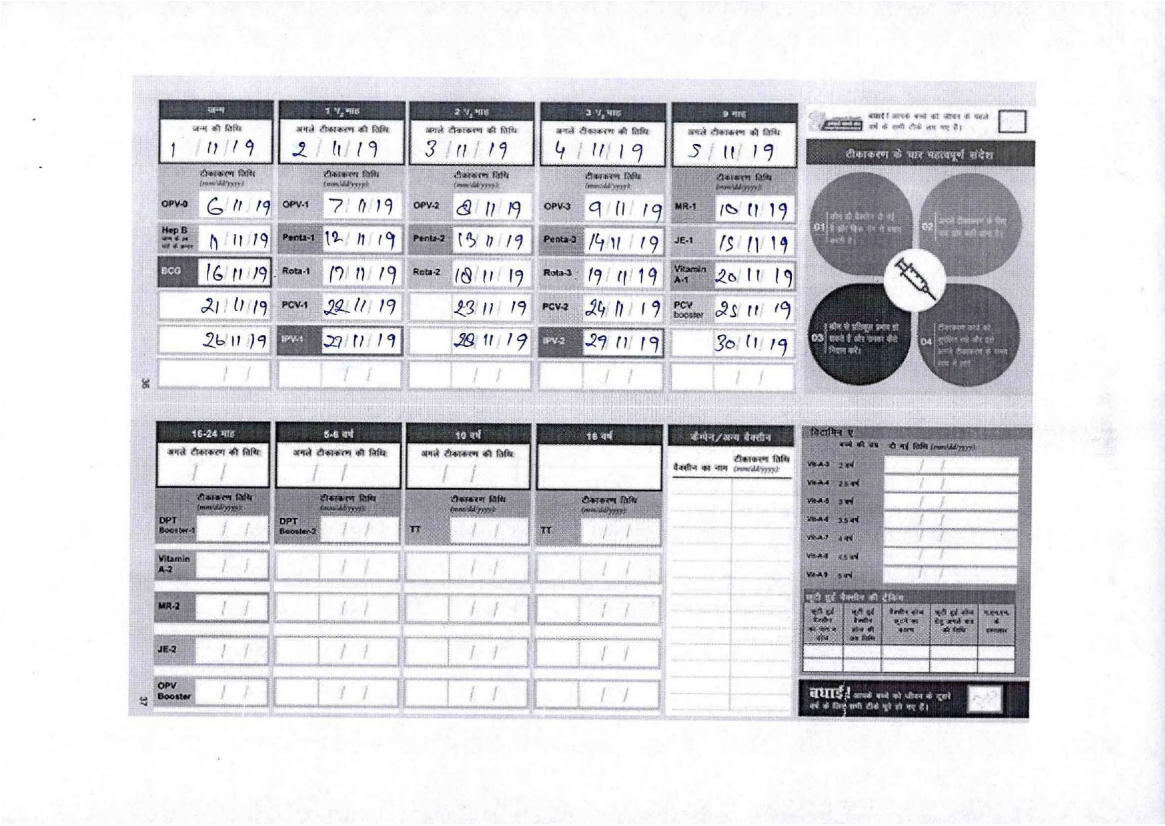

Figure 5: Scanned1

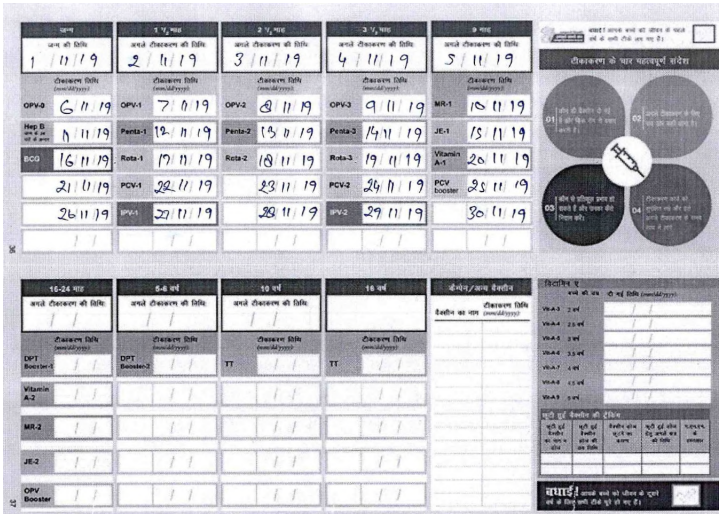

Figure 6: Aligned

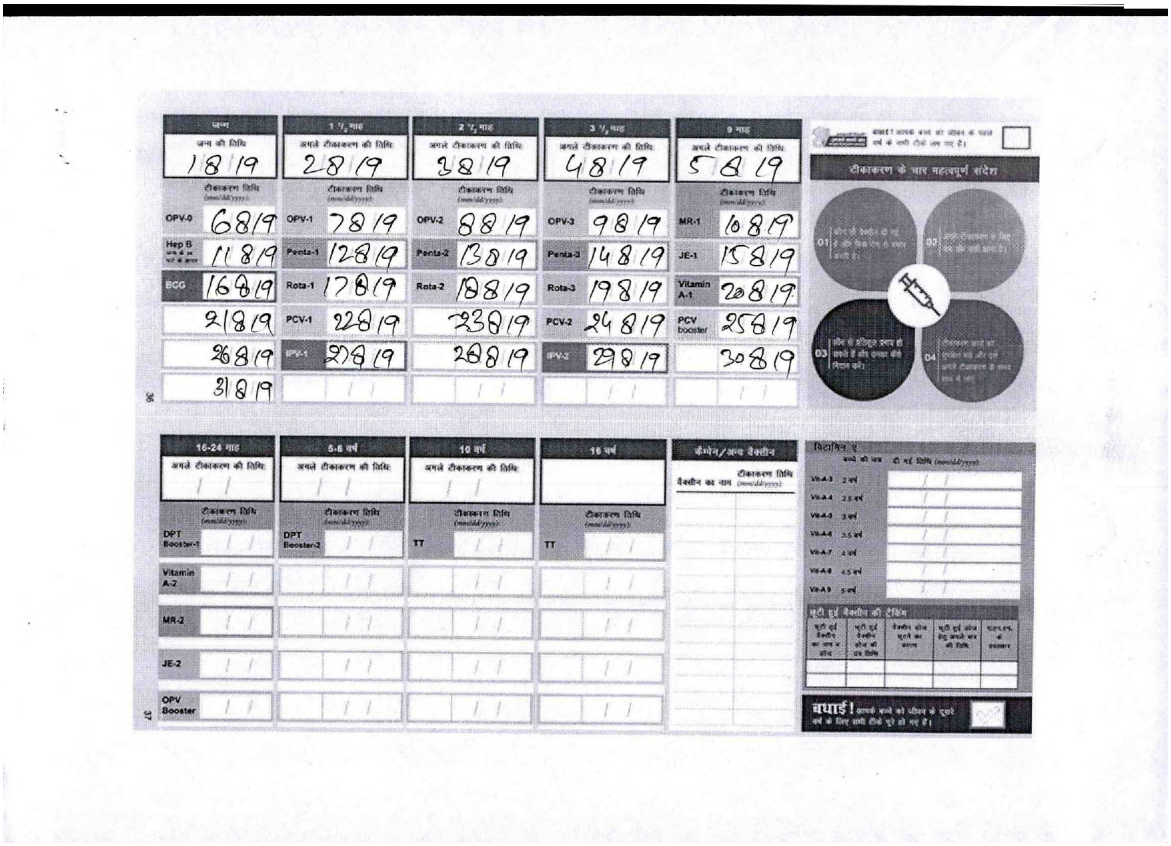

Figure 7: Scanned2

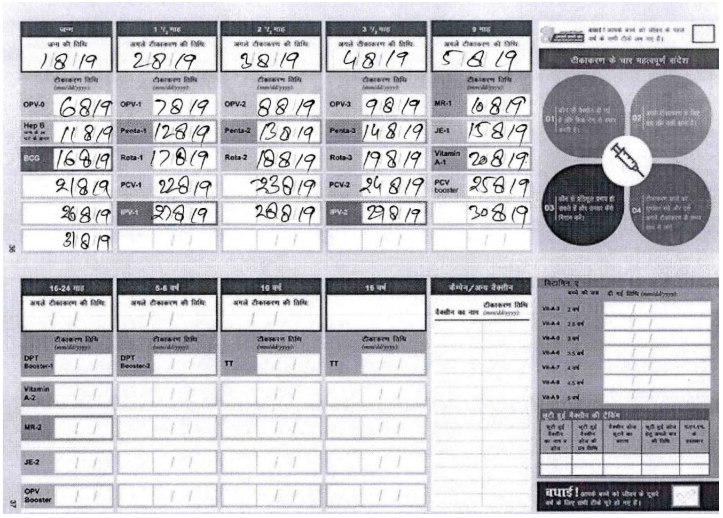

Figure 8: Aligned

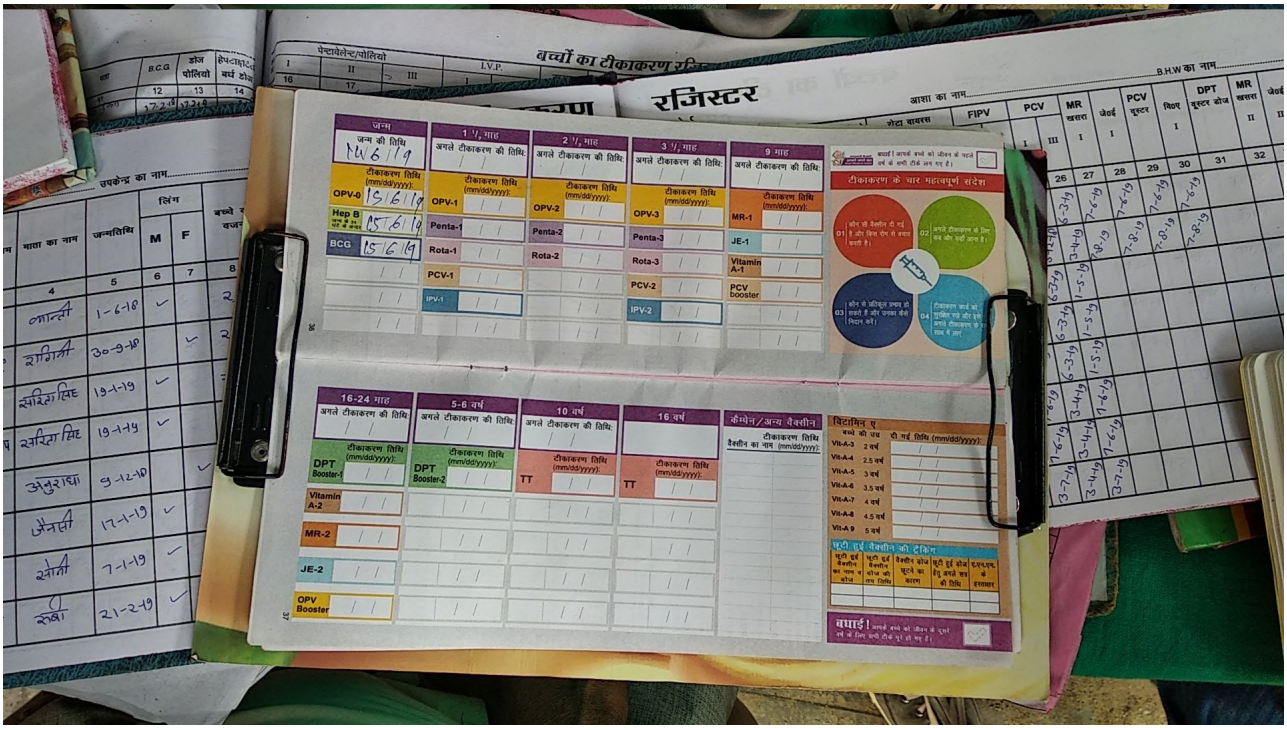

Figure 9: Booklet1

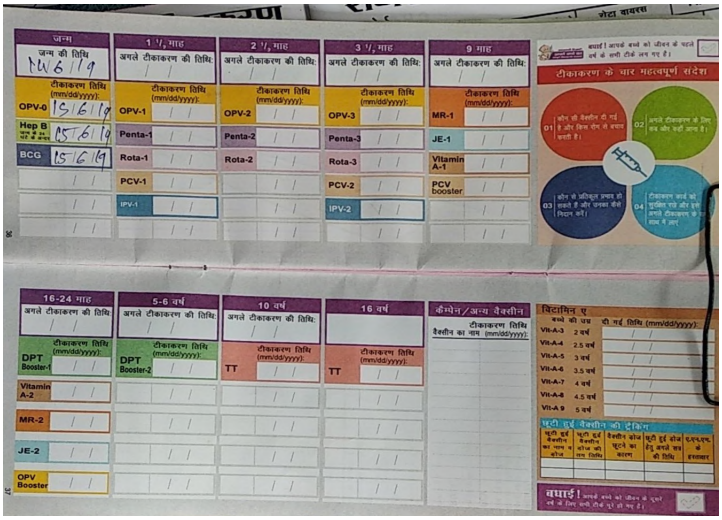

Figure 10: Aligned

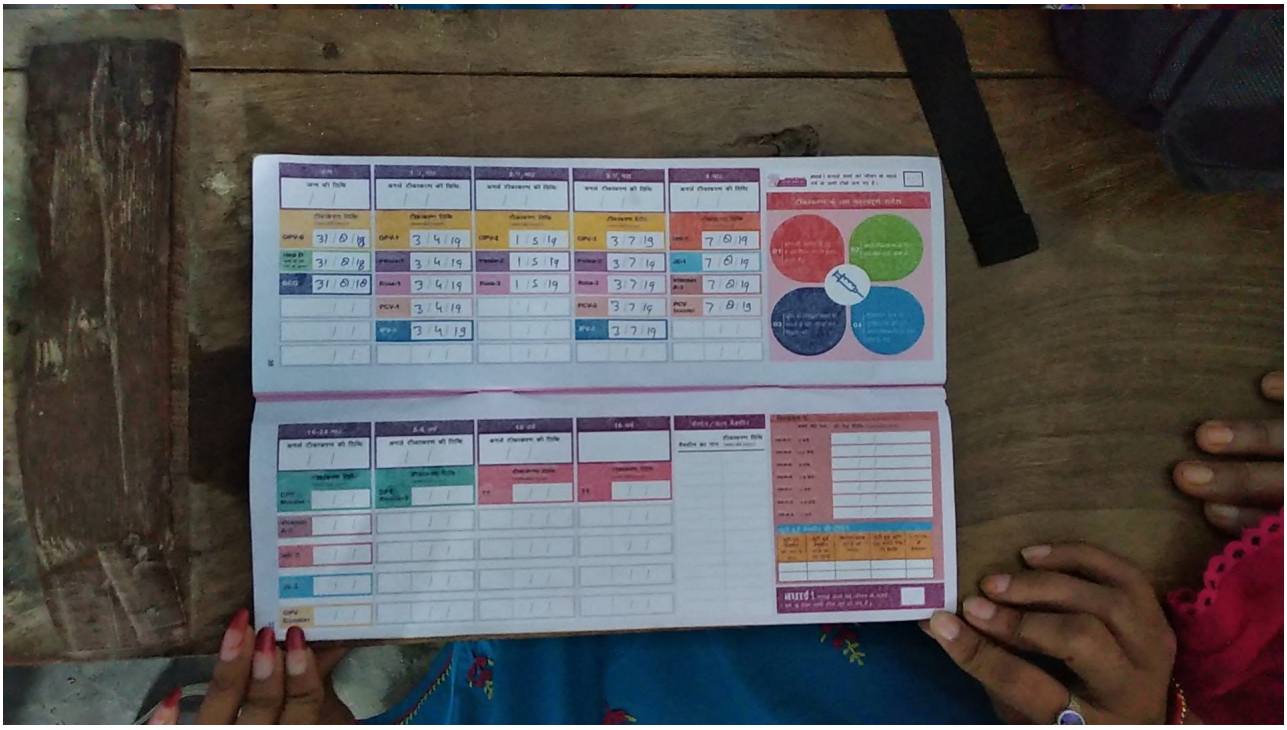

Figure 11: Booklet2

Figure 12: Aligned

Image Segmentation

I tried, various approaches for segmenting the text fields from the complete image. Of these various attempts, I obtained decent results via two approaches, described below.

Contour Map

I implemented `Canny-Edge Detection` on the aligned image, and carved out boxes of desired aspect-ratio to get the boxes. This approach did work, but wasn't complete, i.e. many boxes were left out.

Predefined-Map

Since, the image alignment procedure, tries to warp most of the boxes in the same position, majority of boxes have fixed coordinates. I, thus, worked on a single image and through various measures (contouring) came up with the ideal mask for that image, but that mask can now be applied to all the images. This made segmentation very fast as well as majorly complete! The mask is presented below.

Also, the results for both the procedures are presented below.

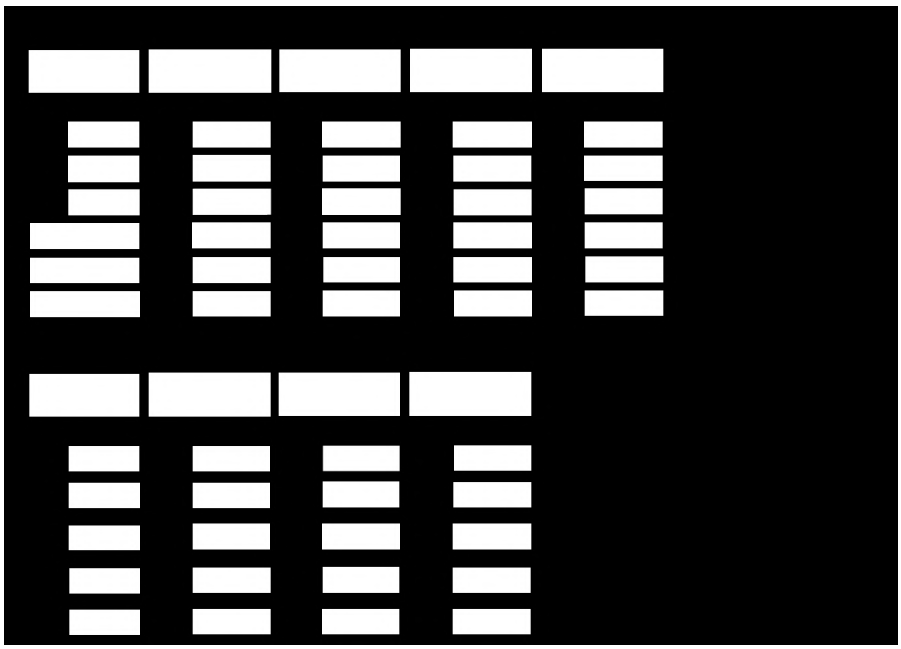

Figure 13: Mask

Contour Map

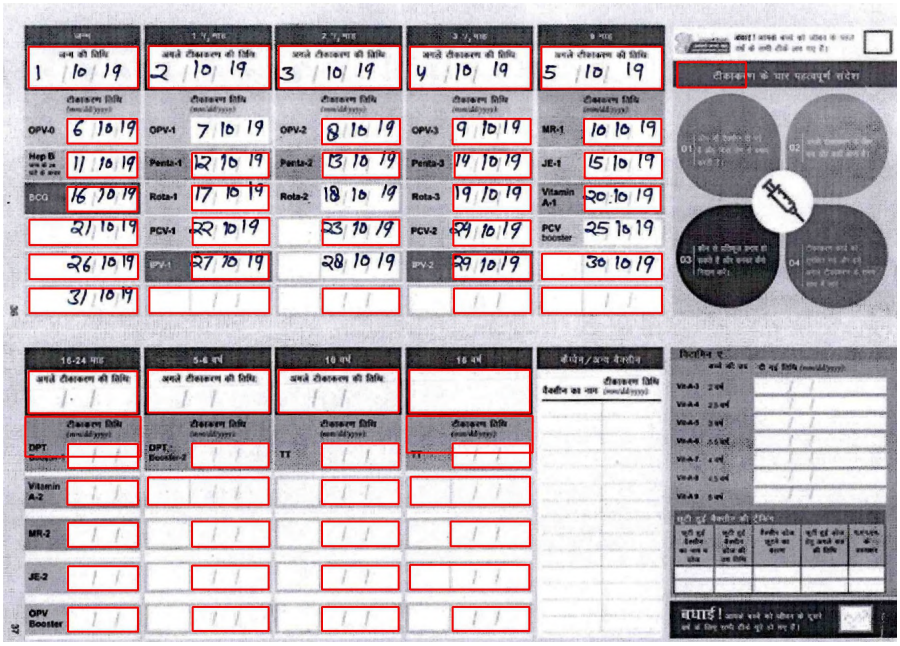

Figure 14: Scanned1

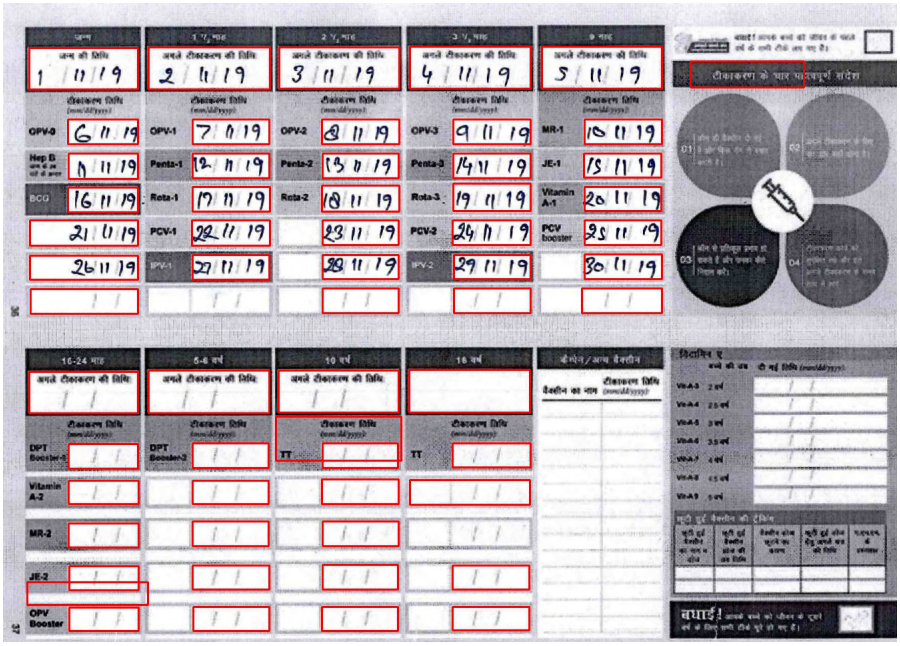

Figure 15: Scanned2

Contour Map

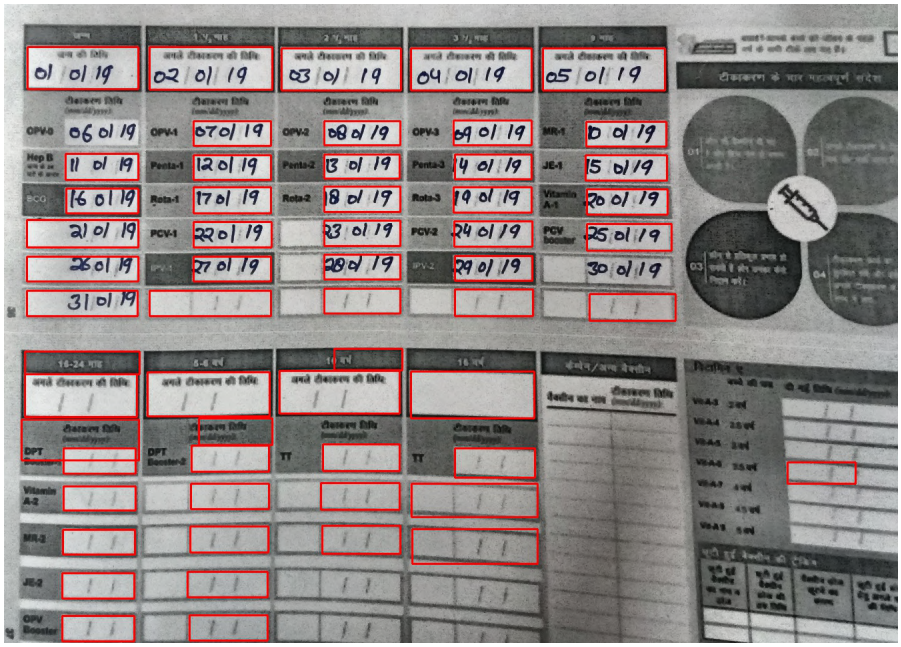

Figure 16: Printout1

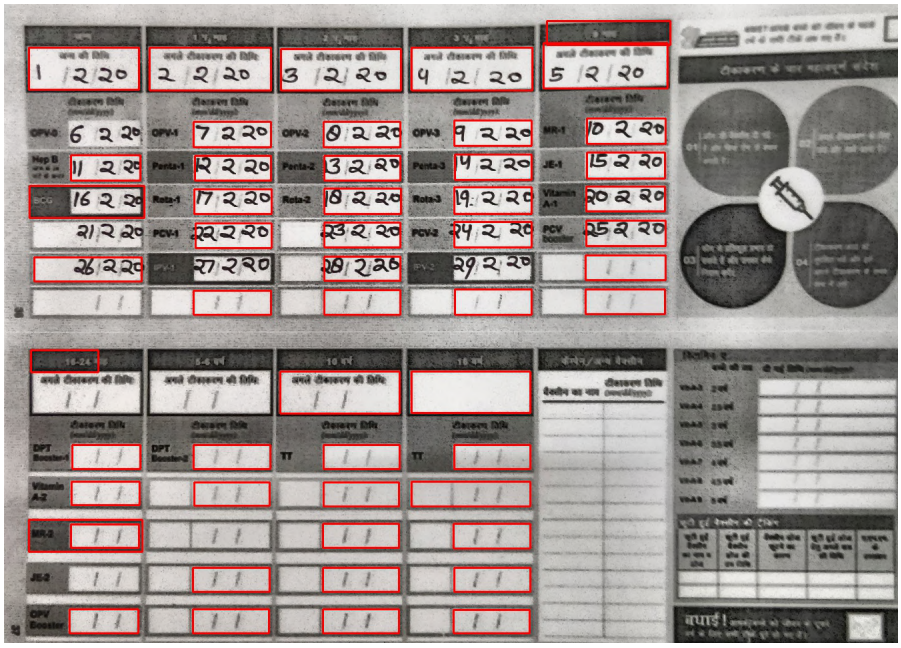

Figure 17: Printout2

Contour Map

Figure 18: Booklet1

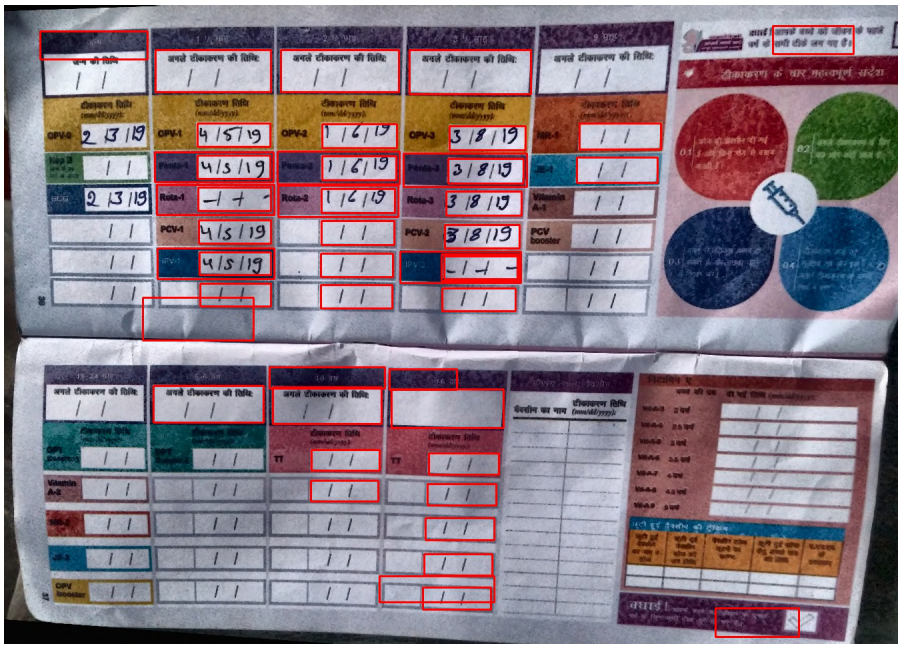

Figure 19: Booklet2

Masked

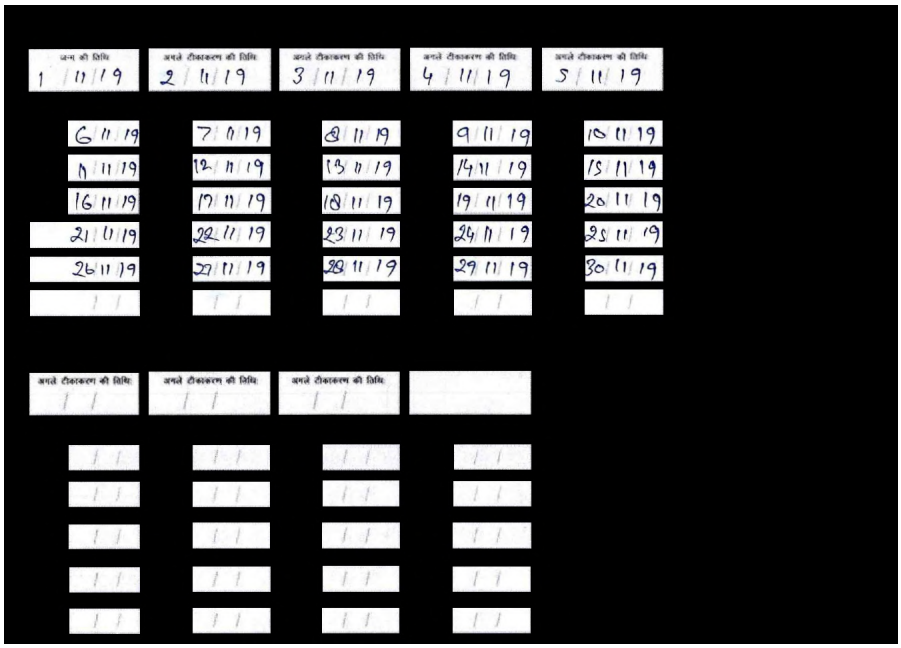

Figure 20: Scanned1

Figure 21: Scanned2

Masked

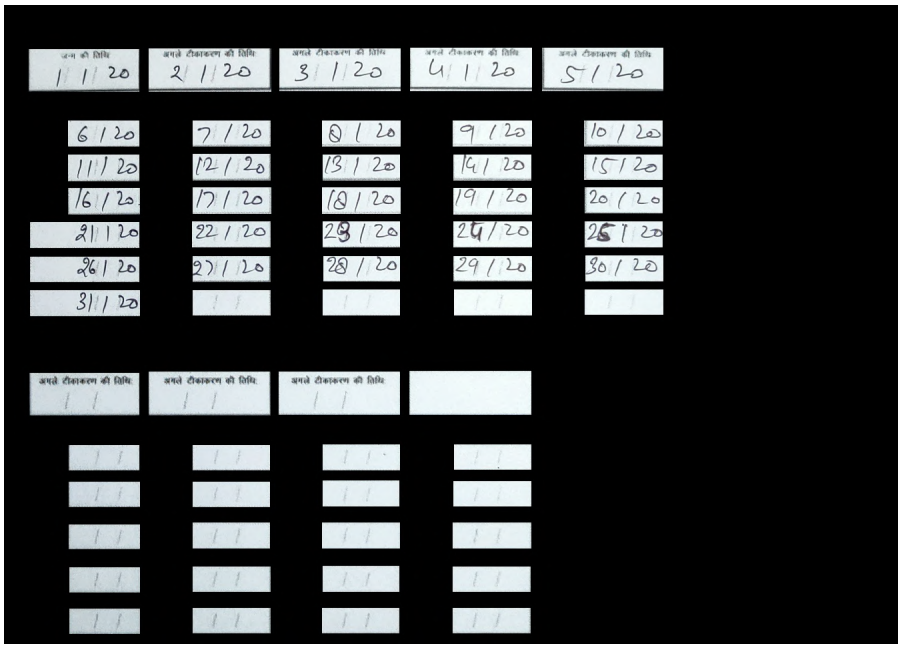

Figure 22: Printout1

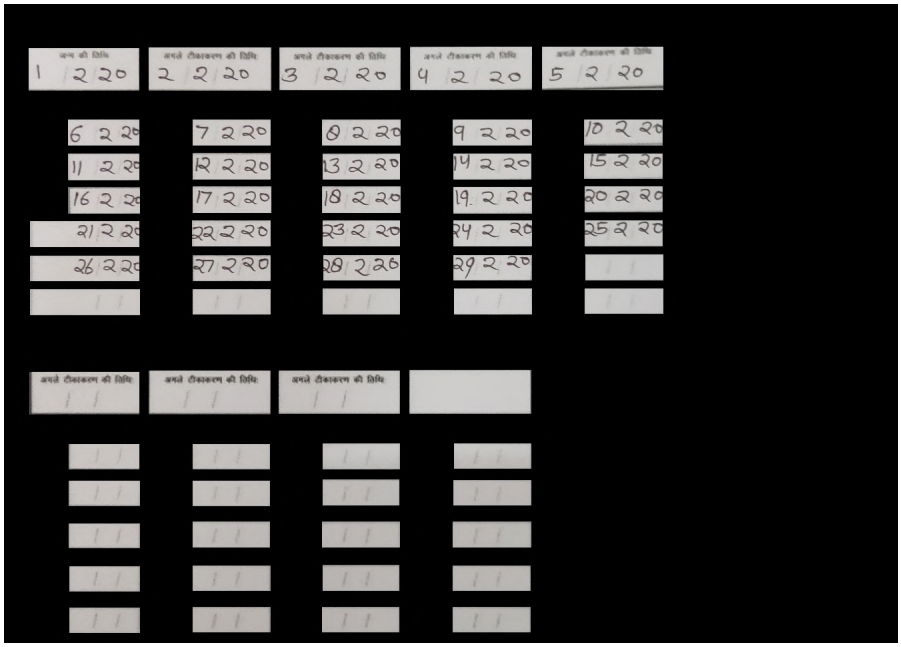

Figure 23: Printout2

Masked

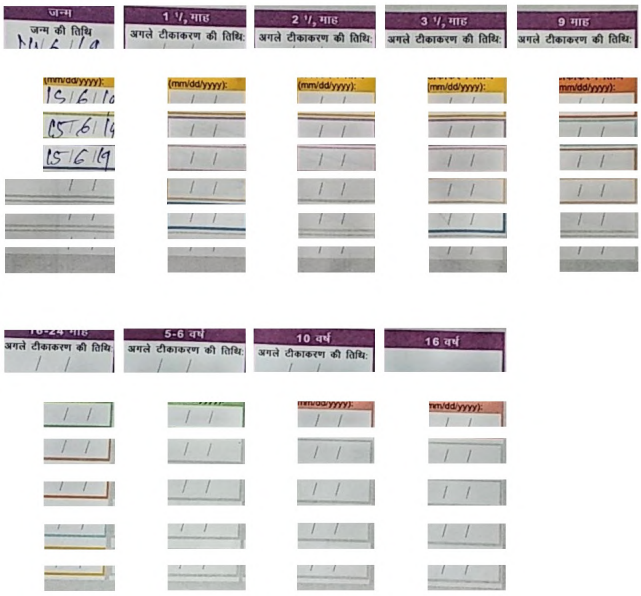

Figure 24: Booklet1

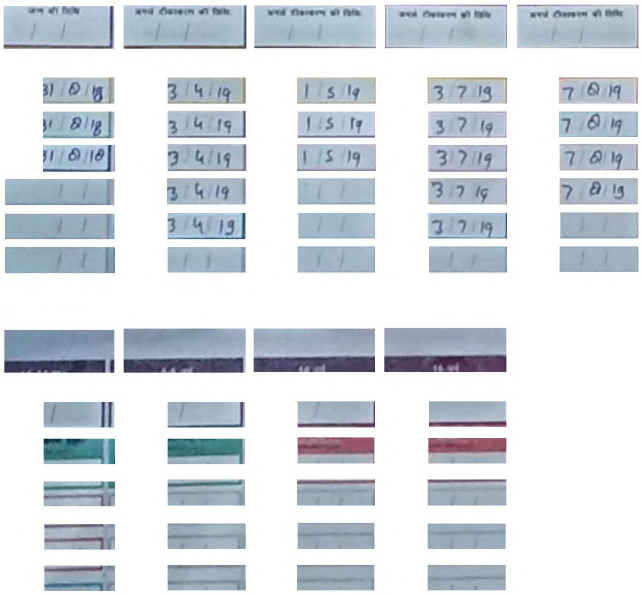

Figure 25: Booklet2

Character detection

Having segmented the image, using the latter approach (masking), it was fairly easy to do character detection using thresholding and then contouring the remaining characters.
Note: The thresholding was binary, after adaptive histogram-equlisationn was applied to the aligned image. This allowed for faster and easier detection.
Bonus: It was observed that median filtering the images led to removal of pre-printed slashes and thus they didn't appear in the detected character-set.
The results, including the detected characters and thresholding results are presented further.

Scanned1

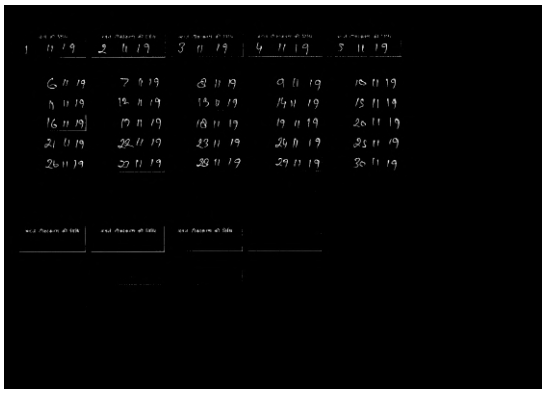

Figure 26: Thresholded Image

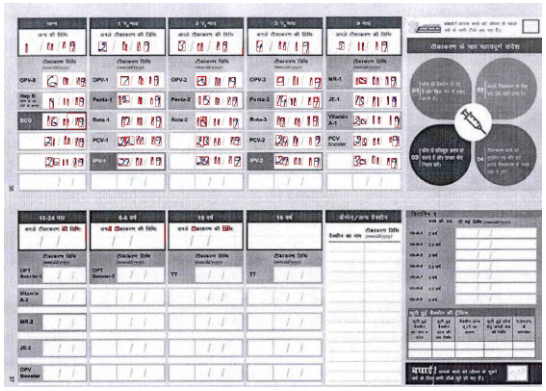

Figure 27: Character Detection

Scanned2

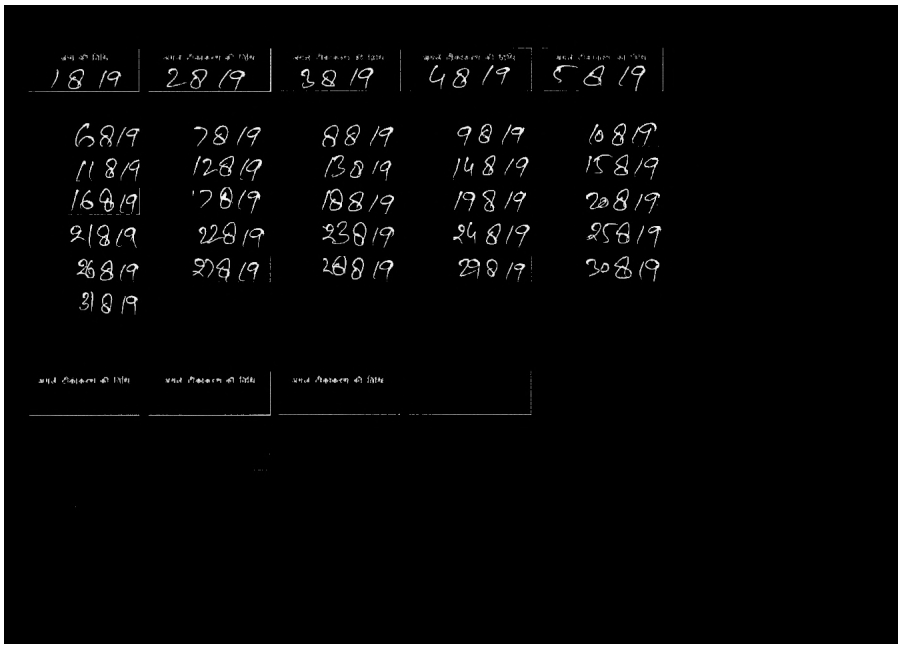

Figure 28: Thresholded Image

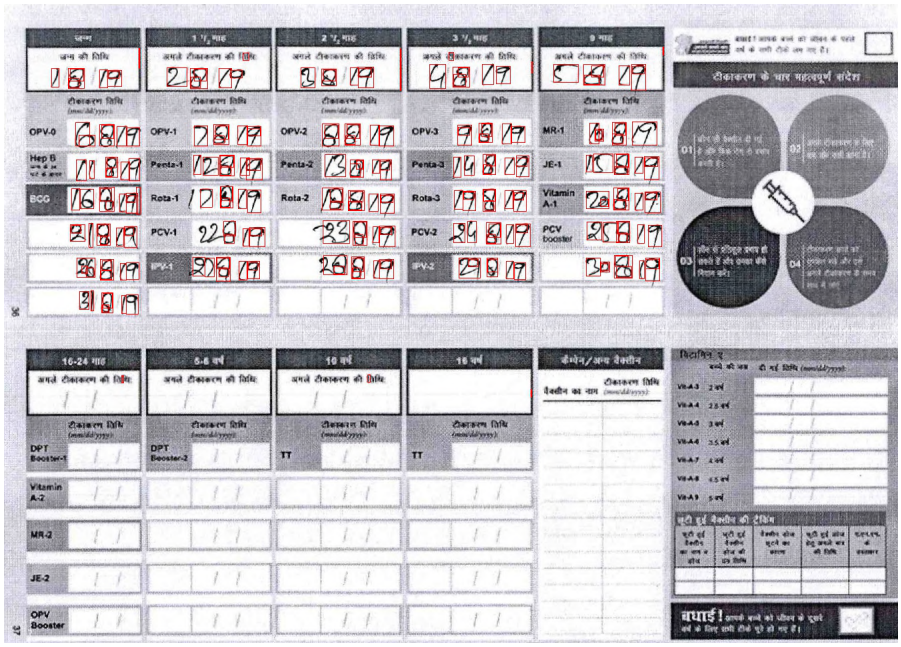

Figure 29: Character Detection

Printout1

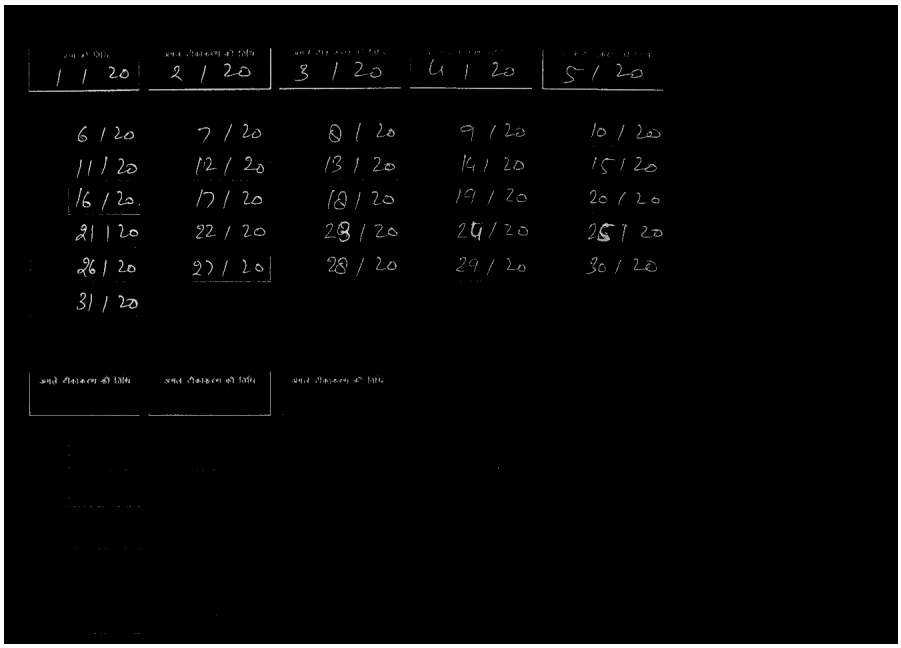

Figure 30: Thresholded Image

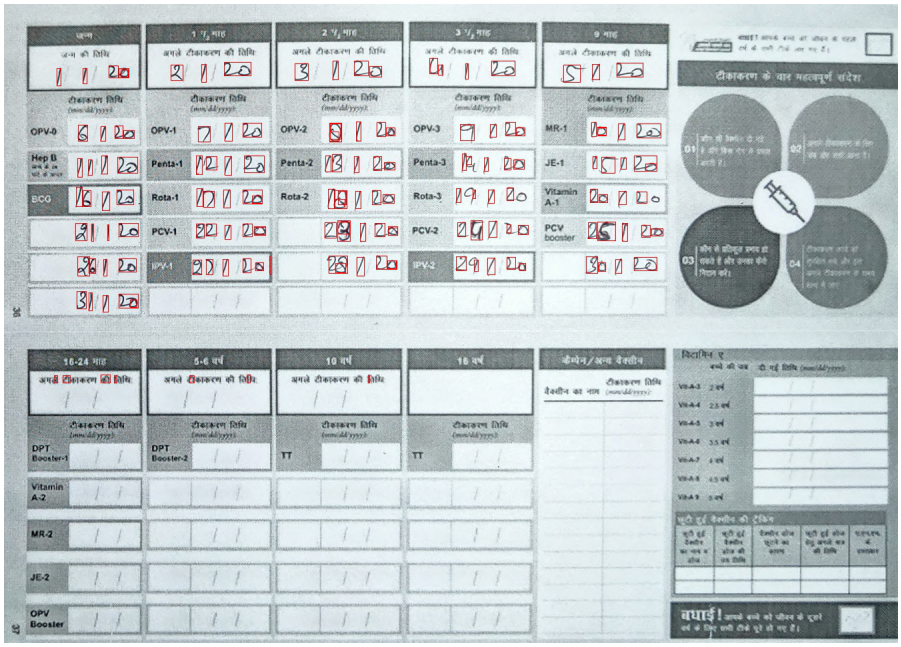

Figure 31: Character Detection

Printout2

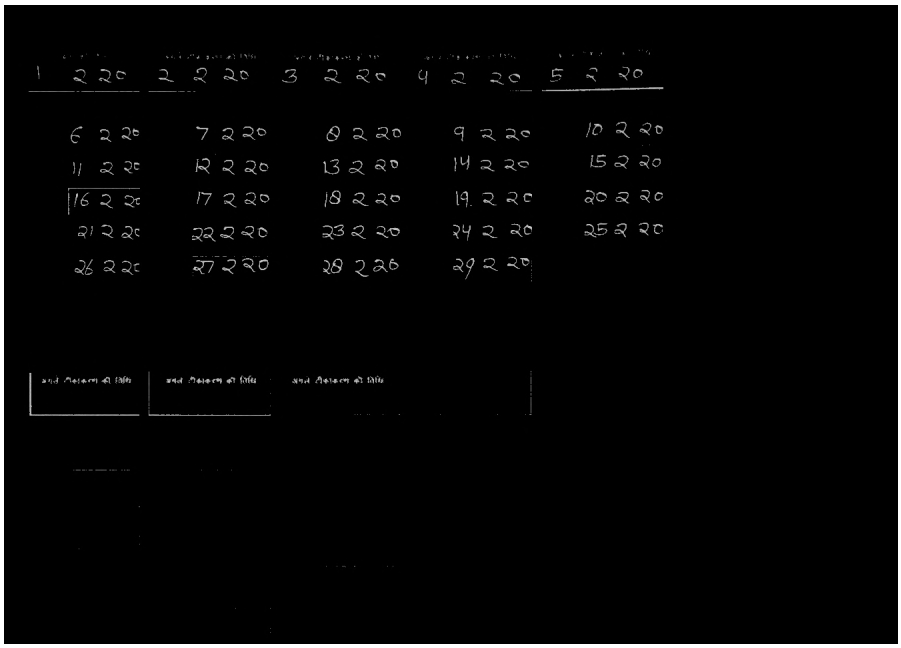

Figure 32: Thresholded Image

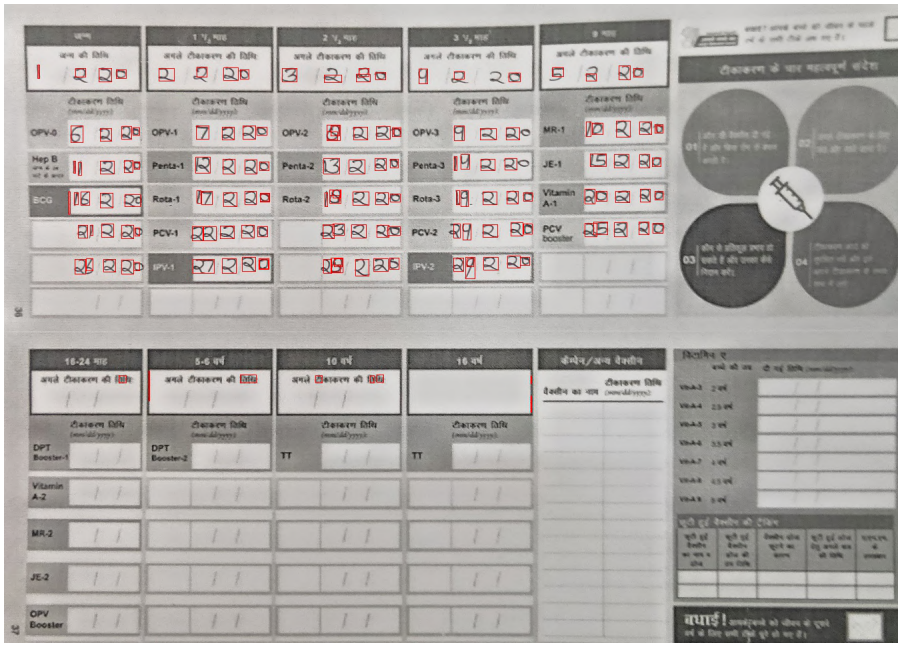

Figure 33: Character Detection

Booklet1

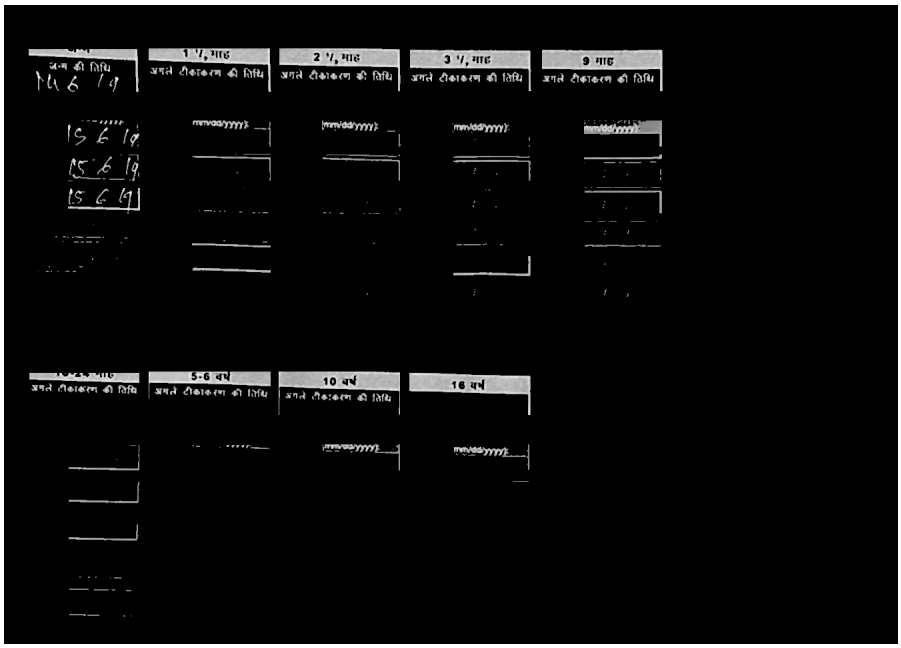

Figure 34: Thresholded Image

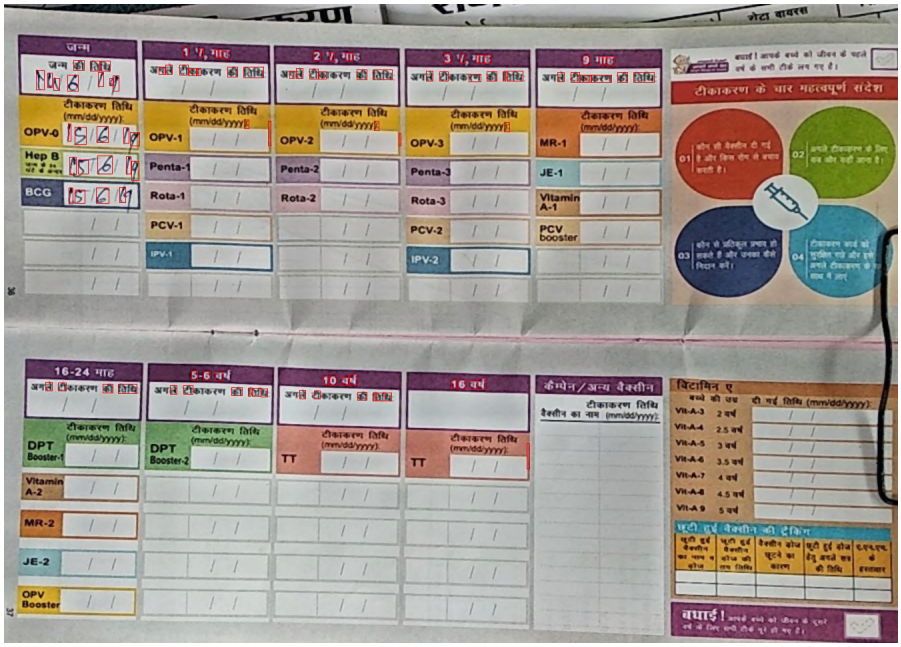

Figure 35: Character Detection

Booklet2

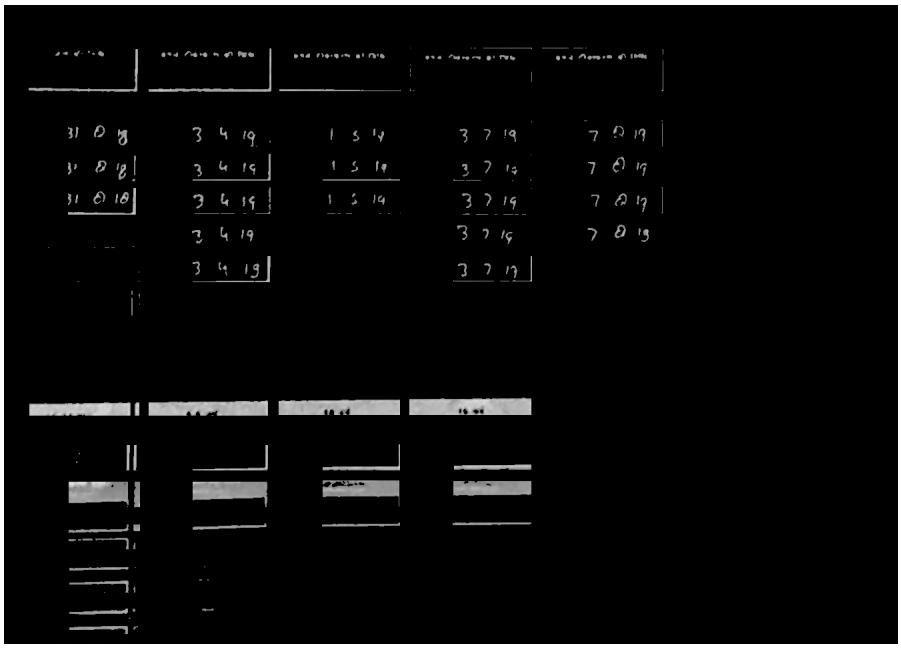

Figure 36: Thresholded Image

Figure 37: Character Detection

Summary

This assignment was very interesting, and I learnt a lot of new things. I really liked the idea of making this assignment open ended. As a direct practical application, we were very motivated to try all the possible things out and see how actually all methods we know of go around. What was discussed in class was very helpful.
Thanks and Regards
Rajbir