HTML Atomic UI Elements Extraction from Hand-Drawn Website Images using Mask-RCNN and novel Multi-Pass Inference Technique

Prasang Gupta and Swayambodha Mohapatra

PricewaterhouseCoopers US Advisory, Mumbai, India
prasang.gupta@pwc.com,swayambodha.mohapatra@pwc.com

Abstract. Website UI Design is an integral part of the world, but it is not trivial as there are a huge array of challenges that need to be conquered. A quintessential step of a website design process is to sketch the UI wireframe on paper and translating it into code later on. In an attempt to automate this process, advanced AI algorithms are explored in this study. The final approach comprises of image processing, followed by UI feature identification and localisation using Mask-RCNN and ultimately a novel Multi-Pass inference technique to boost the viability of the model. On the test dataset, the method resulted in an mAP or Mean Average Precision (IoU > 0.5) value of 64.12

Keywords : HTML, UI, Image Processing, Deep Learning, OpenCV, Mask-RCNN, Multi-Pass Inference

1 Introduction

With the world going increasingly global and starting to work virtually, websites are more important than ever to expand business and reach out to customers. However, website design requires a very specific set of skills. There are 2 major ways of building a website. The first method is using visual website building tools like Wix [3], Constant Contact [1], Squarespace [2], etc. and the second is building by programming using languages like HTML, PHP, CSS, JavaScript, etc. The downside of both of these approaches is that they have a very steep learning curve.

The ImageCLEF 2020 DrawnUI Task [5] from ImageCLEF 2020 [10] is formulated to reduce this dependency on the tools and flatten the learning curve by enabling people to create websites using hand-drawn pictures of website interfaces on whiteboard or a piece of paper. This would give a chance to people having no knowledge of the aforementioned tools and languages to create websites easily and quickly.

Table 1: A list of unique labels present in the data set.

Index	Label
0	paragraph
1	dropdown
2	checkbox
3	radiobutton
4	rating
5	toggle
6	textarea
7	datepicker
8	stepperinput
9	slider
10	video
11	label
12	table
13	list
14	header
15	button
16	image
17	linebreak
18	container
19	link
20	textinput

The first step towards making this possible is to come up with a model that correctly identifies the type and the position of various atomic user interface (UI) elements in the wireframe drawing. This information can be leveraged to generate a website layout using various heuristics. The next step to this problem would be to convert this detected layout to code. In this study, we are focusing on the first part of the problem.

Diving into the details of the implementation, we will discuss about the Dataset used for training the model in Section 2 and will cover the Methodology used in Section 3. Further, we will discuss the Results in Section 4 and present the Conclusions and any scope for Future Work in Section 5.

2 Data Set

ImageCLEF 2020 DrawnUI task [5] was focused on extracting the atomic UI elements from a hand-drawn image of a website. The dataset provided as part of the challenge contained about 3000 hand-drawn images inspired from mobile application screenshots and actual web pages containing about 1000 different templates.

The dataset was divided into two parts, the development set which contained 2363 annotated images and a test set containing 587 images which were not

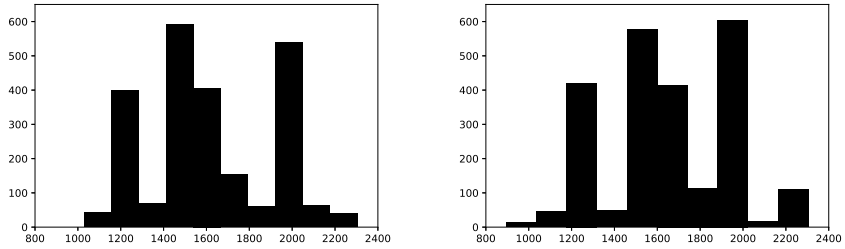

Fig. 1: The plot on the left shows the distribution of the height of the images while the one on the right shows the distribution of the width of the images. Both of these histogram plots have been generated keeping the bin size at 10.

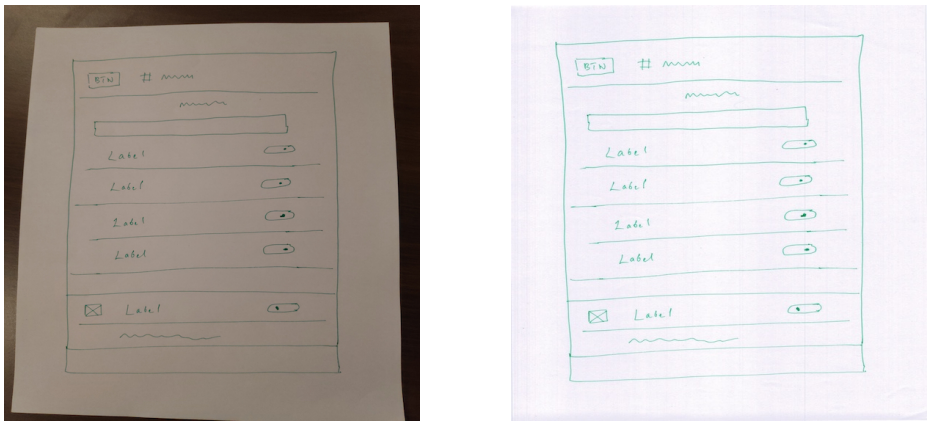

Fig. 2: The image on the left and the right are two distinct images from the development dataset. However, taking a closer look at both of the images, it can be seen that both the images are exactly identical and differ only in the tint and a slight camera angle change.

annotated and were strictly for testing purposes. Each image in the development set contained information in the form of a bounding box and a label for each UI element present in that image. There were 21 different classes of labels present as listed in Table 1.

The images in the dataset were of varying sizes. All of them were RGB images in JPEG format. The annotations were provided separately in a CSV file format. The distribution of the width and the height of the images can be seen in Figure 1. Due to the varying sizes of the images, they need to be resized to a fixed size that will be covered later in the modelling section.

There were several challenges within the dataset. The first challenge was that there were several repeated images in the development set which would not

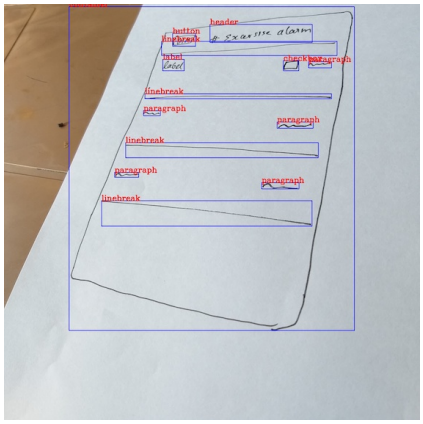

Fig. 3: The image above has been taken at a very steep angle. This converts the straight horizontal lines into diagonals and rectangles into parallelograms.

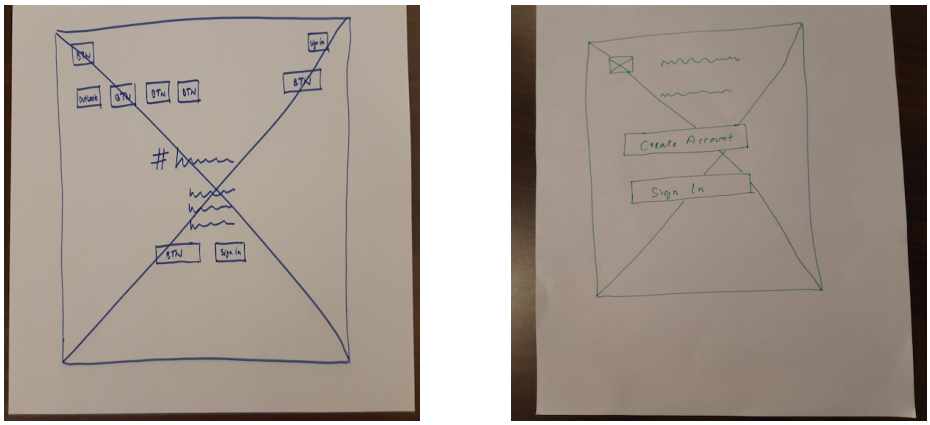

Fig. 4: The images shown above represent the overlap of different classes on the "image" class. There were 2 types of overlap. The image on the left shows the "under the image" overlap, while the image on the right shows the "over the image" overlap. Both of these are taken from the development dataset.

contribute much towards the training of the AI model. These images were not exactly similar, but differed only on the basis of the background colour / tint of the image. One of such examples of this can be seen in Figure 2.

The second challenge was that there were some images with a very steep capture angle in the dataset as shown in Figure 3. This had to be taken care of as the apparent shapes of the UI elements would change drastically when the images are captured at an angle.

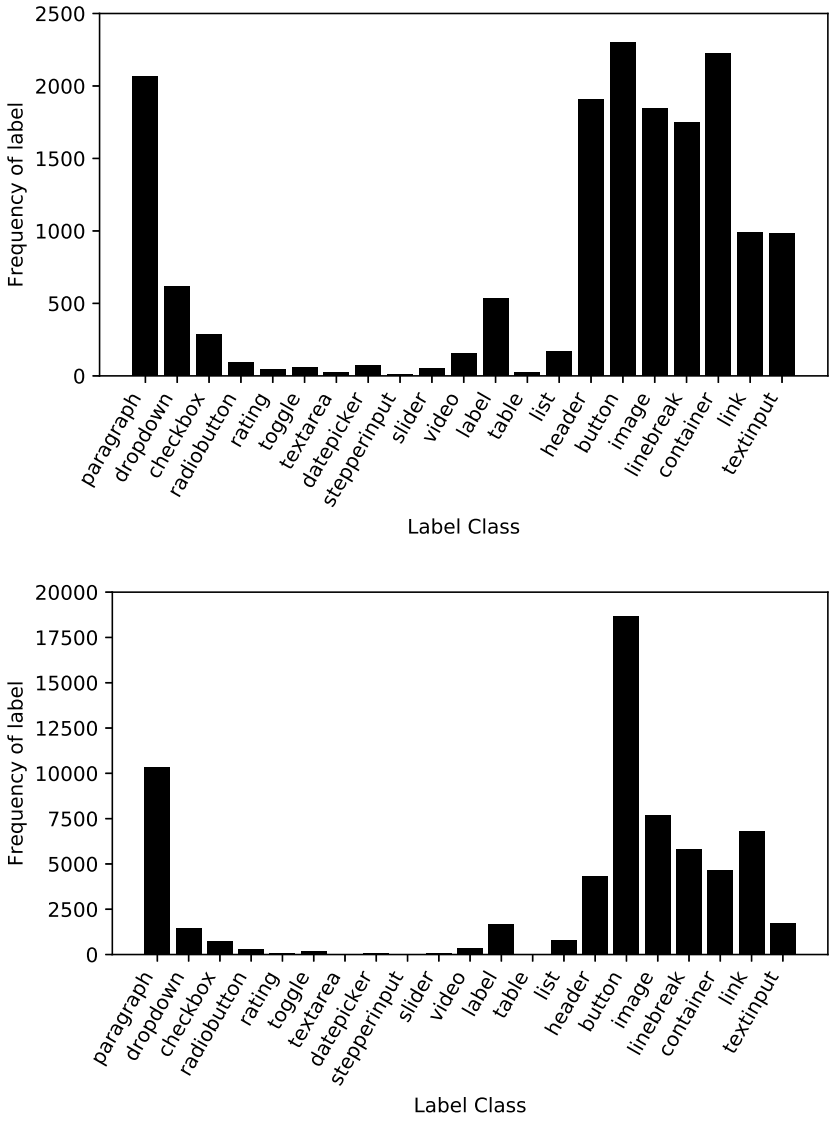

Fig. 5: The plots [9] above show the distribution of various class labels in the dataset. The plot on the top shows the number of images out of the development dataset, in which that label was present. The plot on the bottom shows the total number of occurences of the label across the dataset. As we can see from both the plots, the dataset is very skewed towards certain classes.

The third problem pertained to the ”image” class of the dataset. The image class was defined as a rectangle with both the diagonals drawn. However, there

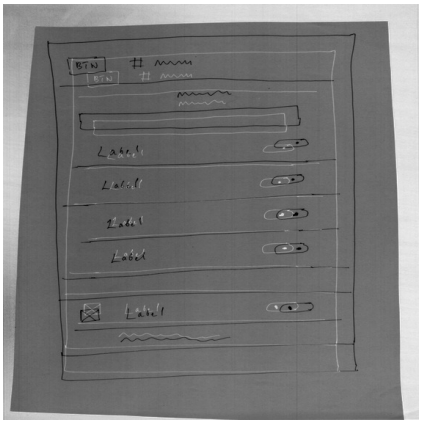

Fig. 6: The image above is the result of the subtract operation applied on Figure 2. It can be observed from the image that due to the shift in the capture angles, the subtract function has not worked as expected.

were several files containing multiple objects overlapping with an image class. There were 2 kinds of overlap, the object over the image, where the diagonal was ”hidden” or behind the image where the wireframe of the image would run over these classes. This can be seen in Figure 4.

Apart from these challenges, the number of labels were also skewed in the dataset as some labels had plenty of representation, while some labels were present quite rarely. The distribution for the labels can be seen in Figure 5. It can be seen from the plot that the labels like ”button”, ”paragraph” and ”image” are very commonly present while other labels like ”textarea”, ”stepperinput” and ”rating” are very sparse.

3 Methodology

3.1 Data Pre-Processing

We have explored several pre-processing techniques to improve the viability of our model. The majority of techniques are based on modifications of the data using OpenCV [4] on C++. This was chosen to reduce the time taken to perform operations and transformations on the images. Some of the techniques used are described in this section.

Removing duplicate images

By visual inspection of the dataset as mentioned in Data Set Section, we found that there were repeated images (as can be seen in Figure 2) and thus, these images would not contribute much to the training of the model. If the

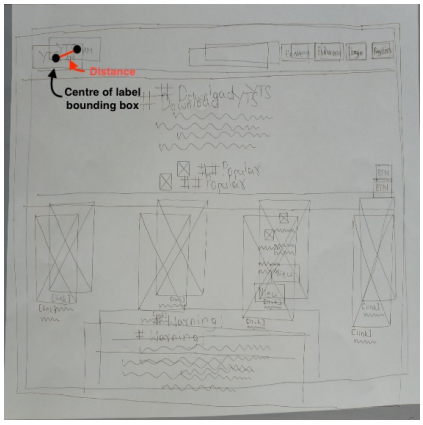

Fig. 7: The image above is the result of the superposition of two candidate images for similarity check. It can be seen that both of the them are exactly similar but a little shifted. To gauge this, the distance between the images can be calculated using the sum of squares of distances between the centre of the corresponding label bounding boxes of both the images. The red line in the image depicts the distance and the black dot is the center of the bounding box for the corresponding labels in both the images

number is huge, then there may be a possibility that our model would start overfitting to these similar images. Hence, to quantify the number of images that are repeated in the dataset, we had to come up with some algorithm for detecting the same. We will be testing the algorithms on the same set of images as shown in Figure 2.

One of the methods commonly used for checking if the images are equal or not is the OpenCV subtract method. This method performs a pixel-by-pixel subtraction of the images and returs an image. If the returned image is completely black, then the starting images are same. We tried employing this method to our images, but the results were not good as can be seen in Figure 6. This can be owed to the fact that our similar images are not "exactly" same as there are some camera angle changes involved which change the orientation of the image and hence, a pixel-by-pixel subtraction did not yield the best results here.

The second approach used for finding duplicate images was an algorithm based on finding the smallest distance between two given images. The algorithm included making a list of size 21 (the total number of unique classes present in the dataset) and then populating it for all the images with the number of the classes of each type they have. This was iterated upon and all the image pairs having the same class vector were found. The cartesian distance between these selected pairs was calculated to verify if the images are actually similar or not and if the distance was found to be lesser than a threshold, it was classified as a repeated image pair. The way of calculating the distance between the two

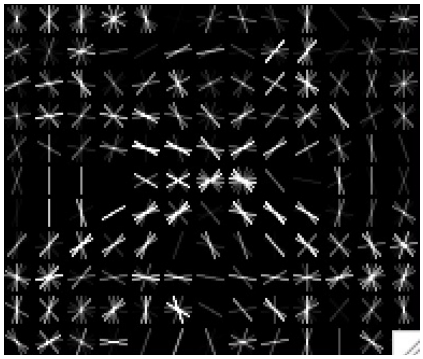

Fig. 8: The image shows the output of the DLIB model trained on just 15 instances. In this very limited learning, it has identified the general structure of the image class with great accuracy.

images can be seen in Figure 7. Employing this algorithm, it was found that there were only 1306 unique images in the development dataset out of the total 2363 images. Rest 1057 images were copies of the images already present.

Extracting individual elements from the image

As the underlying shape of each of the label is same and only the localisation of the labels vary across images, we tried extracting the individual elements from the image. These extractions can then later be used for training purposes. Also, these can be used for increasing the number of the classes whose frequency is less in the dataset. This would allow the model to learn the features of the lesser frequent label types as well.

The approach selected for this was using a DLIB [11] model to capture the general features of the class labels. The DLIB model was chosen as it could learn the basic structure of the class with very few learning data, as would be the case with the classes having very low representation. A sample DLIB model output for the image class can be seen in Figure 8. The DLIB models were able to identify most of the label classes, but were unable to segregate the image wherever there was overlapping present.

Converting images to Grayscale

The images provided in the dataset were all 3 channel RGB images. 3 channels might be helpful in problems where the information carried by the colour is needed to be learned by the model and should be used as a feature. But, in our case, colour doesn't matter as we have to detect the features only on the basis of shape. Hence, to prevent throwing off the learning of the model by introducing

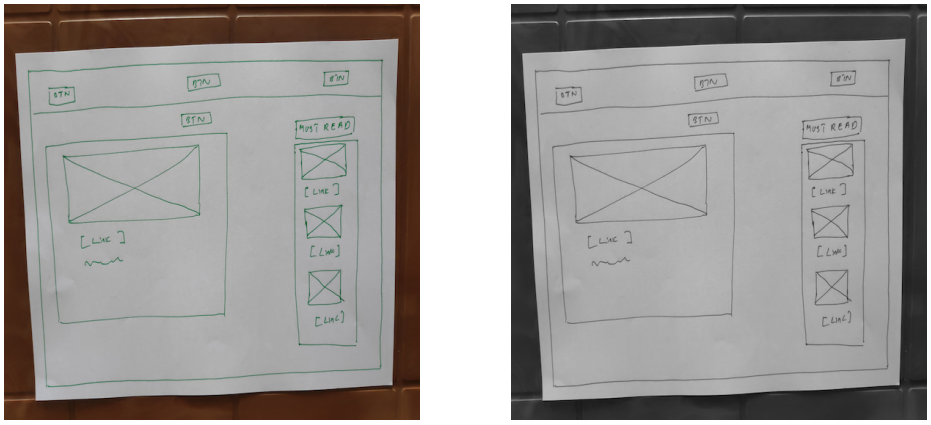

Fig. 9: The image on the left is the original image while the one on the right is after the grayscale conversion.

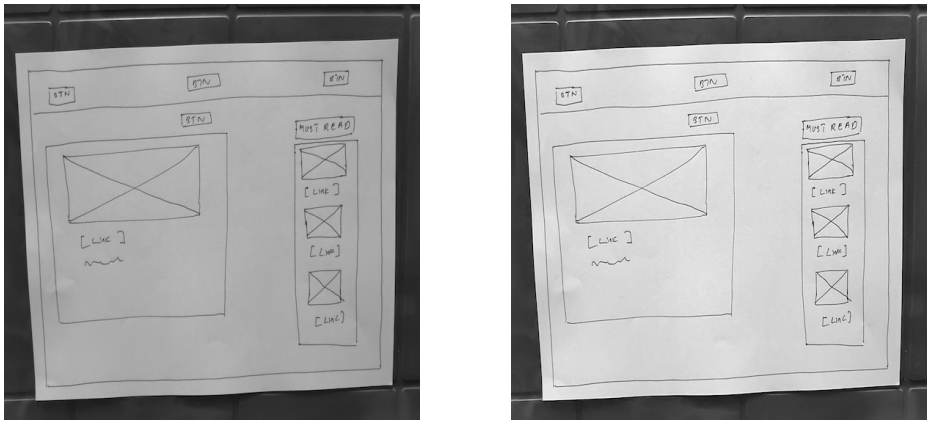

Fig. 10: The image on the left is the converted grayscale image while the one on the right is a more refined sharpened grayscale image.

colour, all the images were converted to grayscale using OpenCV. A sample grayscale conversion can be seen in Figure 9.

Another factor that is helpful in grayscale images is that the number of channels are reduced to 1. Hence, the effective size of the image reduces which leads to a speed up in computation. To increase the visibility of the labels further, the grayscale images were later sharpened using OpenCV. This can be seen in Figure 10.

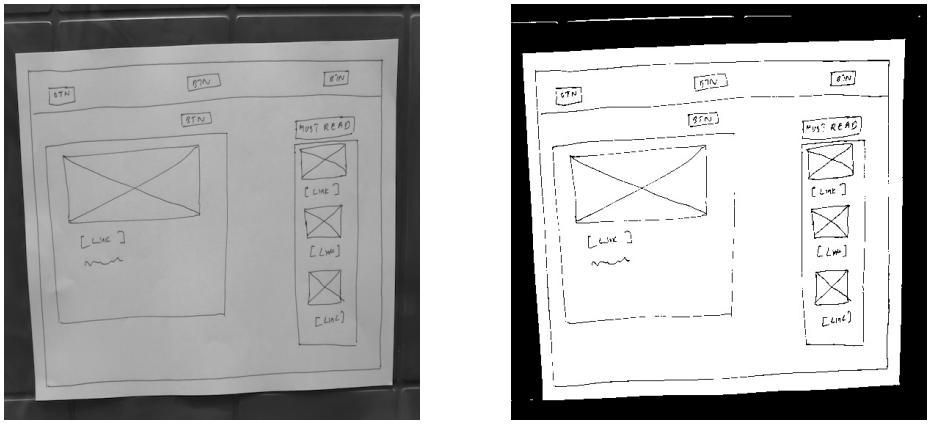

Fig. 11: The image on the left is the grayscale image while the one on the right is the result of a simple thresholding conversion.

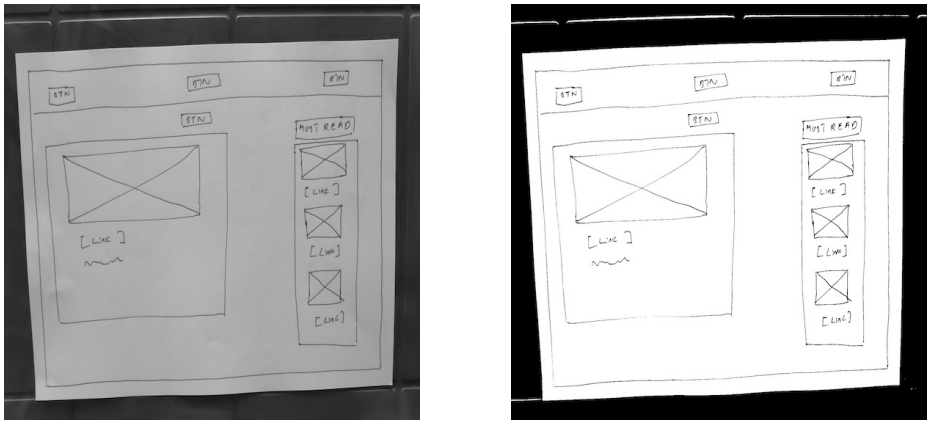

Fig. 12: The image on the left is the grayscale image while the one on the right is after applying Otsu's binarization algorithm on it

Converting images to Black & White

There were several algorithms applied to convert the image from a grayscale image to a Black and White Image. This was carried out to further reduce the effect of the background elements on the model prediction, as grayscale also carries information regarding the shade of the image or background.

The first approach used to convert the grayscale images to black and white was simple binary thresholding. But, the limitation of the model was that every image had a different optimum threshold and there was no way to find it before-

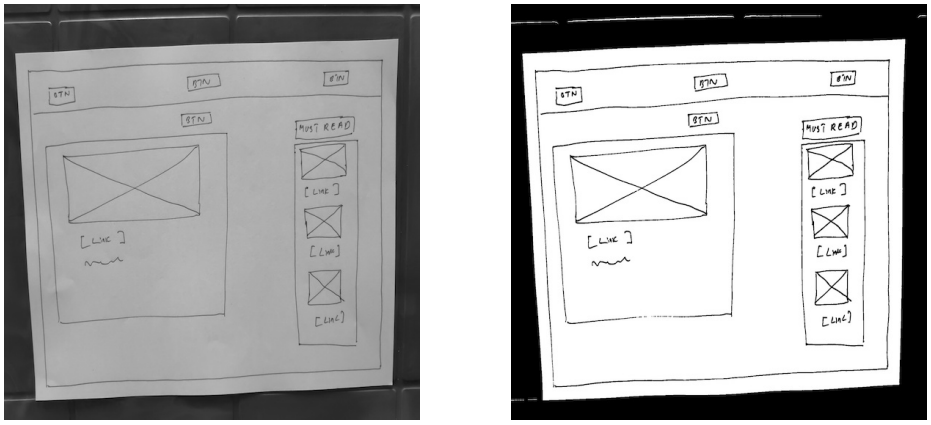

Fig. 13: The image on the left is the grayscale image while the one on the right is after applying erosion on the image and then applying Otsu's binarization algorithm

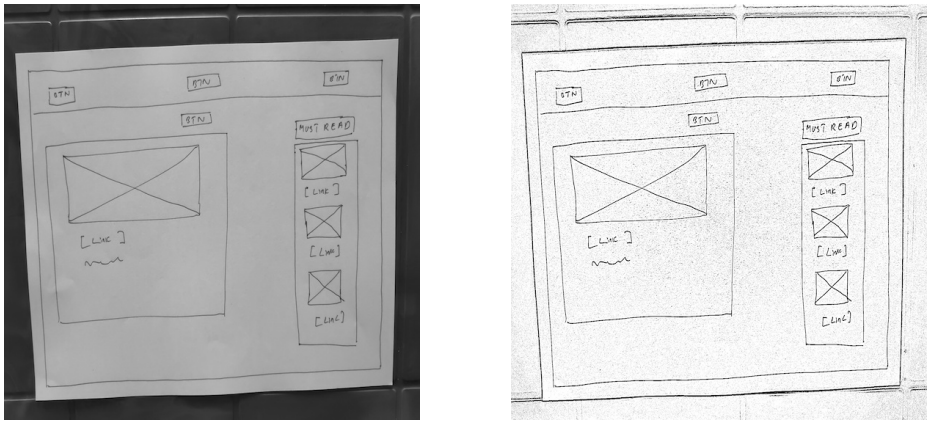

Fig. 14: The image on the left is the grayscale image while the one on the right is after applying adaptive Gaussian thresholding on the image.

hand. Hence, there was a lot of loss of information by this conversion as can be seen in Figure 11.

The second approach used is Otsu's binarization algorithm [6]. The Otsu's algorithm finds a threshold for the image automatically based on its histogram distribution. The image generated using this is shown in Figure 12. Formally, Otsu's algorithm tries to find a threshold value (t) which minimizes the weighted within-class variance given by the following relation.

$$\sigma_w^2(t) = q_1(t)\sigma_1^2(t) + q_2(t)\sigma_2^2(t) \tag{1}$$

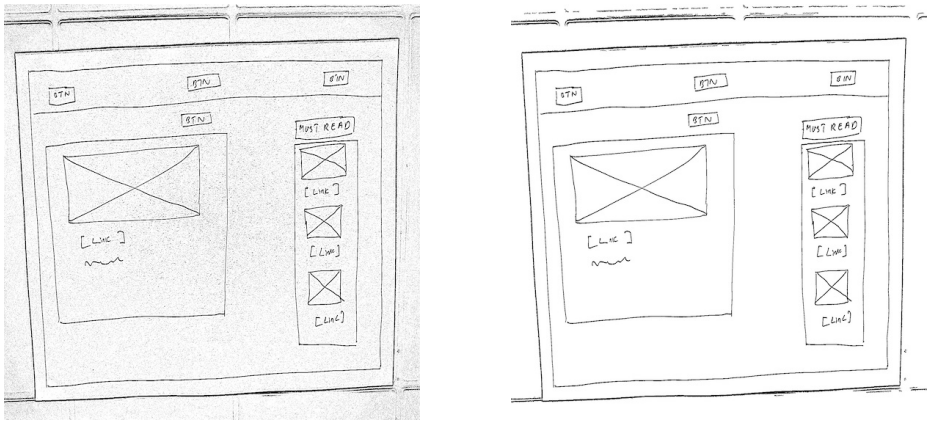

Fig. 15: The image on the left is the image generated after applying adaptive Gaussian thresholding while the one on the right is after running the C++ code to remove the short connected components from the image, reducing noise.

where

$$q_1(t) = \sum_{i=1}^{t} P(i) \;\&\; q_2(t) = \sum_{i=t+1}^{I} P(i) \tag{2}$$

$$\mu_1(t) = \sum_{i=1}^{t} \frac{iP(i)}{q_1(t)} \;\&\; \mu_2(t) = \sum_{i=t+1}^{I} \frac{iP(i)}{q_2(t)} \tag{3}$$

$$\sigma_1^2(t) = \sum_{i=1}^{t} [i - \mu_1(t)]^2 \frac{P(i)}{q_1(t)} \;\&\; \sigma_2^2(t) = \sum_{i=t+1}^{I} [i - \mu_2(t)]^2 \frac{P(i)}{q_2(t)} \tag{4}$$

As an extension of the Otsu's approach, the next approach first Eroded the image to sharpen all the edges and remove connected elements and then applied Otsu on top of it. The outcome for this can be seen in Figure 13.

The final approach used is an adaptive approach where a single threshold is not applied globally to the dataset. The threshold value in this case is a Gaussian-weighted sum of the neighbourhood values minus a constant. The results using this were good, but contained a lot of noise as can be seen in Figure 14. However, this was by far the best conversion of grayscale to binary black and white. Hence, this was selected as the final model and the noise was tackled by finding and removing the small connected components of the image using C++. The final image can be seen in Figure 15.

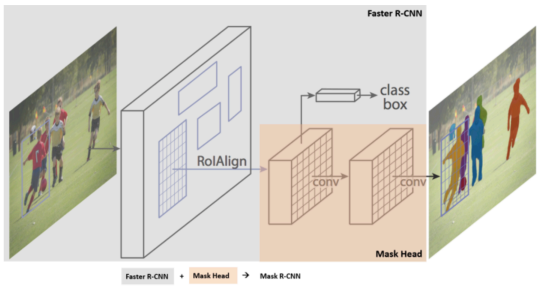

Fig. 16: General Architecture of the Mask RCNN Model. Reproduced from Mask-RCNN [7].

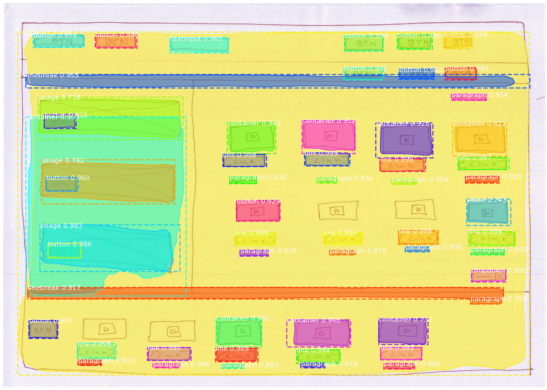

Fig. 17: Output generated by Mask RCNN Model (Run 1) on one of the images belonging to the test split of the dataset.

3.2 Methods Implemented

The images were first all transformed into single channel black and white images and then resized into 1024*1024*1, as required by Mask RCNN Architecture [7]. The general architecture of the Mask RCNN model can be found in Figure 16. The dataset was split in the ratio 80:20 with the larger ratio corresponding to the training set and the smaller one corresponding to the validation set. Since there were a few classes that had a small set of images corresponding to them, care was taken to ensure that such images were present in the same ratio while splitting the dataset. The models were trained on a virtual Ubuntu server equipped with a 16GB NVIDIA Tesla V100 GPU Accelerator [14] hosted on PwC's proprietary cloud platform, Workbench.

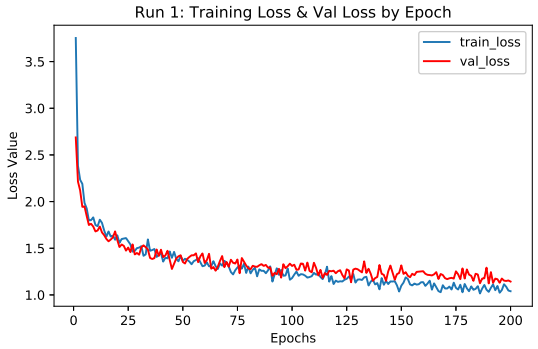

Fig. 18: Graph showing the convergence of Training and Validation Loss of the model used in Run 1

Run 1 : Plain Mask RCNN Model

Since this challenge mainly involved detection of small website UI elements, Mask RCNN was chosen because it performs better than other models in object detection. Mask RCNN model generates bounding boxes and segmentation masks for each instance of an object in the image, and is based on Feature Pyramid Network (FPN) [12] and a ResNet-101 backbone [8].

We implemented Transfer Learning by using a pre-trained Mask RCNN model trained on COCO Dataset [13]. Even though the images contained in the COCO Dataset are not very similar to our dataset, we used it to ensure that our model extracts the high-level features in all images. The 'heads' layer of the model was then trained for 200 epochs at a Learning Rate of 10^{-3}. The convergence of the model can be seen in Figure 18.

The output from this model, obtained on an image from the test split of the dataset can be seen in Figure 17. The model performed really well in recognising all the major UI elements in the image. But, the problem was that it was not able to detect smaller UI elements in the image.

Run 2 : Mask RCNN Model with novel Multi-Pass Inference Technique

After evaluation, even though the overall precision score of Run 1 was high, the overall recall score was not as high. This meant that the model was not able to detect the smaller UI elements on the image. To improve the previous run, we implemented a novel Multi-Pass Inference Technique.

The novel Multi-Pass Inference Technique involves getting the predictions on the input image and then filling the corresponding bounding box regions with the background colour (white in this case). The edited image is then passed

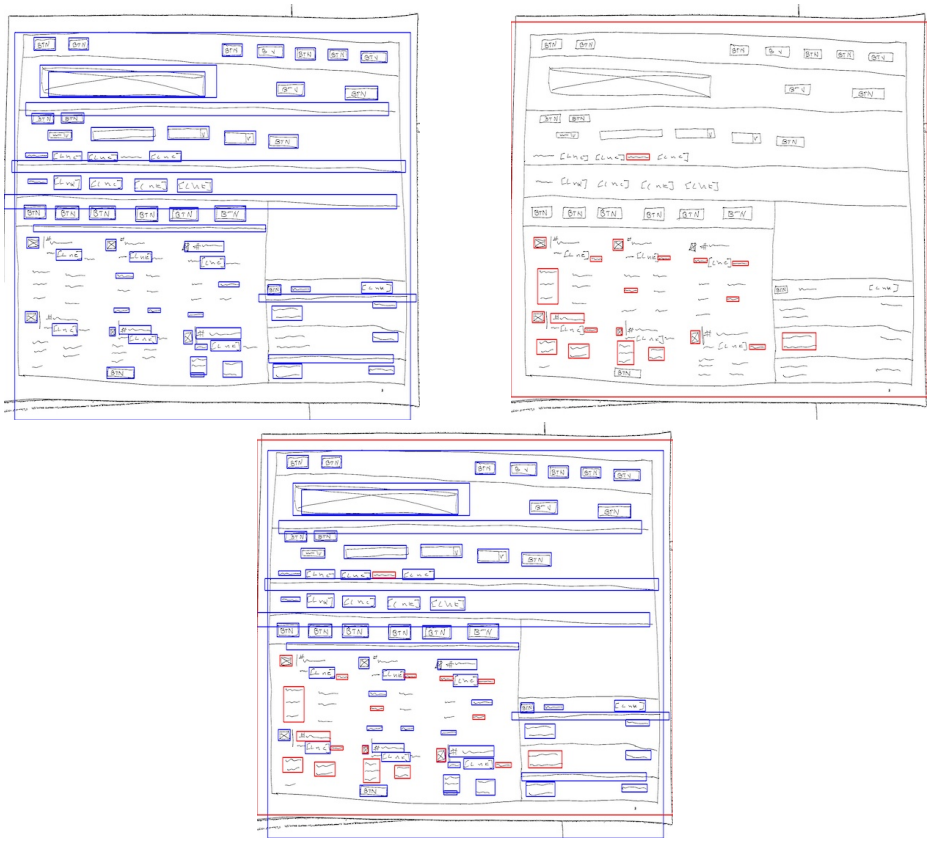

Fig. 19: The image on the top left is obtained after passing the image through the model once (Generally used single pass inference). The bounding boxes in this image were filled with white (except the classes where overlapping is happening) and then passed through the model again. The image on the top right is the output of the 2nd pass. Both of the above images are combined together on the basis of confidence scores and IoU overlaps to form the final prediction image on the bottom. The blue bounding boxes in the final image are from Pass 1 while the red boxes are from Pass 2.

again through the model to essentially 'force' the model to make predictions on the missed out elements. The new predictions are appended to the earlier predictions to get the final results for a particular image. This technique can be visualised in Figure 19. It can be observed from the figure that there are several UI elements that have been missed in Part 1, but predicted in Part 2 making the final output contain most of the UI elements present on the page. The number of times the edited image is passed can be varied according to the problem in hand.

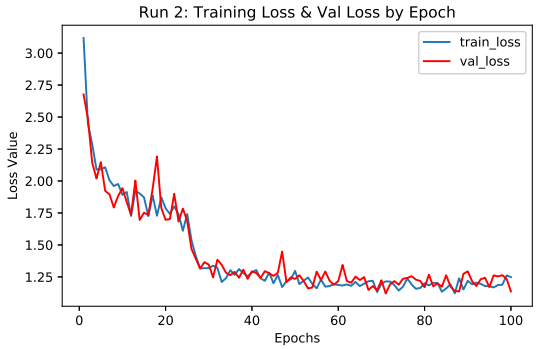

Fig. 20: Graph showing the convergence of Training and Validation Loss of the model used in Run 2

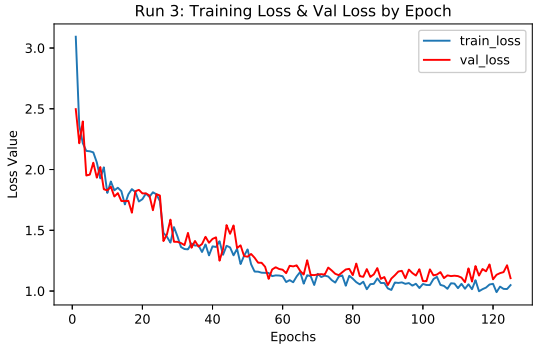

Fig. 21: Graph showing the convergence of Training and Validation Loss of the model used in Run 3

For this particular run, the 'heads' layer of the model was trained for 100 epochs and the following Learning Scheduler was used to ensure the model converges quickly - Learning Rate of 10^{-2} for the first 25 epochs, Learning Rate of 10^{-3} for the next 25 epochs, Learning Rate of 10^{-4} for the next 25 epochs, Learning Rate of 10^{-5} for the last 25 epochs. The convergence of the model can be seen in Figure 20.

Run 3 : Modified Version of Run-2

This run was implemented to improve the results obtained from the previous run. The same Multi-Pass Inference technique was implemented with a slight modification to ensure that only the bounding boxes with the highest confidence scores after the second pass were added to the final results of each image.

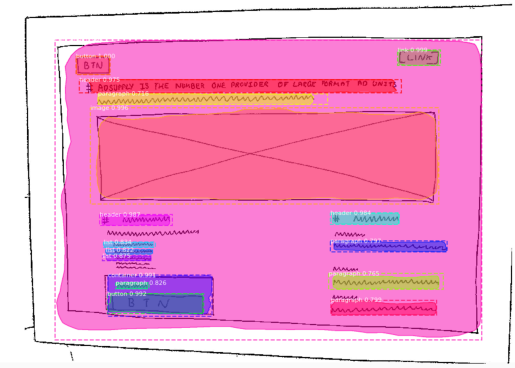

Fig. 22: This image shows the intermediate output generated by Mask RCNN Model with Multi-Pass Inference Technique (Run 3) on one of the images belonging to the test split of the dataset after the first pass. A few smaller UI elements are missed out by the model.

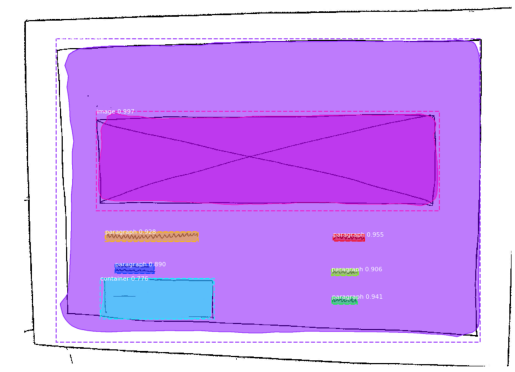

Fig. 23: This image shows the final output generated by Mask RCNN Model with Multi-Pass Inference Technique (Run 3) on one of the images belonging to the test split of the dataset after the second pass. Most of the missed elements in the first pass are captured in the second pass.

This was done to ensure that the stray elements detected after the white space replacement step are not added to the final results.

For this particular run, the 'heads' layer of the model was trained for 125 epochs and the following Learning Scheduler was used to ensure the model converges quickly - Learning Rate of 10^{-2} for the first 25 epochs, Learning Rate of $5 * 10^{-3}$ for the next 25 epochs, Learning Rate of 10^{-3} for the next 25 epochs, Learning Rate of $2 * 10^{-4}$ for the next 25 epochs, Learning Rate of 10^{-4} for the last 25 epochs. The convergence of the model can be seen in Figure 21.

Table 2: Table showing Mean Average Precision over IoU >0.5 (mAP), Overall Precision (OP) and Overall Recall (OR) scores obtained across all runs (with Run IDs as mentioned on the challenge website)

	Run ID	Model Description	mAP	OP	OR
Run 1	67391	Baseline Mask RCNN	57.34	94.04	41.7
Run 2	67699	Mask RCNN with Multi-Pass Inference Technique	63.73	91.81	50.1
Run 3	67712	**Modified Version of Run 2**	**64.12**	91.71	49.6

The intermediate output and final output generated from this model, on an image from the test split of the dataset can be seen in Figure 22 and 23. There was a visible improvement in recognising smaller UI elements on the image which was also reflected in better Mean Average Precision (mAP) scores as listed in the Table 2.

4 Results

The predictions on the test set images were collated in a csv file. For each image on the test set, the bounding boxes corresponding to each instance of a detected class and the confidence scores were submitted. The Mean Average Precision (mAP) scores obtained across the three runs can be found as listed in Table 2.

5 Conclusions and Future Work

Throughout the challenge, we experimented with several processing techniques to get the data in the best shape to be trained. We selected Mask R-CNN as our baseline model as it is known to perform well on Object Detection problems and this problem statement was not much different. We also came up with a novel technique, Multi-Pass Inference, which improved the mAP score drastically, hence gaining us the 3rd spot on the leaderboard of the DrawnUI challenge.

Due to the lack of time, we could not tinker around much with the models as the training takes up a lot of time, being computationally expensive. In the future, we can explore other models as the baseline model which have better performance over Mask R-CNN. One such example of an improved model would be EfficientDet, which is known to perform much better, but is deadly slow to train. Also, there is a lot of scope in expanding the viability of the novel Multi-Pass Inference technique and study the affect of number of passes with performance. There is also scope for experimenting with attention mechanism to focus on those parts of the image which are actually important.

References

1. Constant contact. https://www.constantcontact.com/website/ (July 2020)
2. Squarespace. https://www.squarespace.com (July 2020)
3. Wix. https://www.wix.com (July 2020)
4. Bradski, G.: The OpenCV Library. Dr. Dobb's Journal of Software Tools (2000)
5. Fichou, D., Berari, R., Brie, P., Dogariu, M., Ştefan, L.D., Constantin, M.G., Ionescu, B.: Overview of ImageCLEFdrawnUI 2020: The Detection and Recognition of Hand Drawn Website UIs Task. In: CLEF2020 Working Notes. CEUR Workshop Proceedings, CEUR-WS.org <http://ceur-ws.org>, Thessaloniki, Greece (September 22-25 2020)
6. Gonzalez, R.C., Woods, R.E., Eddins, S.L.: Digital image processing using MATLAB. Pearson Education India (2004)
7. He, K., Gkioxari, G., Dollár, P., Girshick, R.: Mask r-cnn. In: 2017 IEEE International Conference on Computer Vision (ICCV). pp. 2980–2988 (2017)
8. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition (2015)
9. Hunter, J.D.: Matplotlib: A 2d graphics environment. Computing in Science & Engineering **9**(3), 90–95 (2007). https://doi.org/10.1109/MCSE.2007.55
10. Ionescu, B., Müller, H., Péteri, R., Abacha, A.B., Datla, V., Hasan, S.A., Demner-Fushman, D., Kozlovski, S., Liauchuk, V., Cid, Y.D., Kovalev, V., Pelka, O., Friedrich, C.M., de Herrera, A.G.S., Ninh, V.T., Le, T.K., Zhou, L., Piras, L., Riegler, M., l Halvorsen, P., Tran, M.T., Lux, M., Gurrin, C., Dang-Nguyen, D.T., Chamberlain, J., Clark, A., Campello, A., Fichou, D., Berari, R., Brie, P., Dogariu, M., Ştefan, L.D., Constantin, M.G.: Overview of the ImageCLEF 2020: Multimedia retrieval in lifelogging, medical, nature, and internet applications. In: Experimental IR Meets Multilinguality, Multimodality, and Interaction. Proceedings of the 11th International Conference of the CLEF Association (CLEF 2020), vol. 12260. LNCS Lecture Notes in Computer Science, Springer, Thessaloniki, Greece (September 22-25 2020)
11. King, D.E.: Dlib-ml: A machine learning toolkit. Journal of Machine Learning Research **10**, 1755–1758 (2009)
12. Lin, T., Dollár, P., Girshick, R., He, K., Hariharan, B., Belongie, S.: Feature pyramid networks for object detection. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 936–944 (2017)
13. Lin, T.Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C.L., Dollár, P.: Microsoft coco: Common objects in context (2014)
14. Lindholm, E., Nickolls, J., Oberman, S., Montrym, J.: Nvidia tesla: A unified graphics and computing architecture. IEEE Micro **28**(2), 39–55 (Mar 2008). https://doi.org/10.1109/MM.2008.31, https://doi.org/10.1109/MM.2008.31