Deep Learning for UI Element Detection: DrawnUI 2020

Naveen Narayanan*, Nitin Nikamanth Appiah Balaji*, and Kavya Jaganathan

Sri Sivasubramaniya Nadar College Of Engineering
{naveen17097, nitinnikamanth17099, kavya17074}@cse.ssn.edu.in

Abstract. As the field of software development is rapidly growing, it becomes vital to show steady progress in the application development sector to maintain the pace. Constantly tweaking the front-end of an application becomes cumbersome for developers. Detection tools that help design UI from hand-drawn sketches can aid developers and non-developers in building applications with great ease. The recent advancements in deep learning techniques and the availability of computational power facilitates training efficient models for object detection. Two such models, the YOLOv4 and Cascade RCNN are implemented for detecting UI elements from hand-drawn sketches and are detailed in this paper. These models are observed to have an average mAP@IoU 0.5 improvement of 31.85% over the baseline Faster RCNN.

Keywords: DrawnUI · YOLOv4 · Cascade RCNN · Object Detection.

1 Introduction

With recent needs in the software development industry for tools to increase efficiency of application development, aiding tools like recommendation systems, generative models help in reducing the time taken to build quality systems. With the difficulty in iterative software engineering setups, it becomes cumbersome for UI designers and front-end developers to make changes and fix issues. So the requirement of an interactive tool for designers to work on, for generating previews or even generate a completely finished front-end application becomes convenient.

As the field of computer vision and deep learning have attained maturity, various network architectures have been built for numerous downstream applications like object detection, instance segmentation, semantic segmentation and panoptic segmentation. The rise of CNN's and its huge success in image classification tasks have encouraged further research on CNN based models for object detection.

* These authors contributed equally

In this paper we discuss about various object detection models and their performance comparison for UI element detection.The rest of the paper is divided into 5 sections. Section 2 gives an idea of the evolution of object detection models and the motivation behind the chosen networks. Section 3 gives an overview of the dataset used. Section 4 explains the chosen architectures and Section 5 compares the performance of the proposed methods and Section 6 draws the conclusions.

2 Background

Object Detection using deep neural networks(DNN) has been an active area of research for over a decade. R-CNN, proposed by Ross Girshick et al. [5] had a two stage architecture, which combined a proposal detector and a region-wise classifier that became predominant in the recent past due to its success. Subsequently SPP net, proposed by Kaiming He et al. [6] improved over RCNN with correct estimation and detection efficiency in testing as analysed by [13]. Regardless of region proposal generation used in all the above networks, the training of all network layers can be processed in a single-stage with a multi-task loss. This was implemented in Fast RCNN [13][4] which saved the additional expense on storage space,and improved both accuracy and efficiency with more reasonable training schemes. It was later found that the Faster RCNN [11] can overcome the region proposal computational cost by implementing a RPN module becoming a significant improvement over Fast RCNN. Cascade RCNN [2], which is a multi stage extension of Faster RCNN, achieves high quality object detection by effectively rejecting close false positives. This is achieved by combining a cascaded bounding box regression and cascaded detection which simultaneously increases both, the quality of hypotheses and the detector.

YOLO, developed by Joseph Redmon et al. [8], had a different approach when compared to R-CNN, where-in a single network predicted both, the bounding boxes and the class probabilities for these boxes. The YOLOv1 divided input images into Size x Size grid cells and in each grid, certain number of bounding boxes were taken. Bounding boxes are then selected if their class probabilities are more than a particular threshold value. But YOLOv1 had difficulties in detecting small objects that appear in groups and in detecting objects having unusual aspect ratios. Combating this issue, YOLOv2 as described by [9], used batch normalisation, a high resolution classifier, anchor boxes and dimension clusters to improve the performance. It also bounds the location using logistic activation overcoming the instability of YOLOv1 in early iterations. YOLOv2 also uses Darknet-19 thus obtaining a good balance between accuracy and model complexity. The next proposed YOLOv3 had few incremental improvements on YOLOv2. It had a better feature extractor, DarkNet-53 with shortcut connections as well as a better object detector with feature map upsampling and concatenation as mentioned in[10]. The latest YOLOv4 has inbuilt Data Augmentation for a more robust training as part of its 'Bag of Freebies' and it improves accuracy of object detection using its 'Bag of Specials' as explained in [1] on the backbone and Detector.

In our work, we thus implement Cascade RCNN and Yolov4 in detecting UI elements because of their significant advantages and state-of-the-art performances.

3 Dataset Description

The dataset provided by [3] [7], consists of hand drawn images of internet websites, mobile application interfaces from 1000 different templates. The dataset consisted of 21 classes which included a variety of small elements like check boxes and buttons all the way to large elements such as images and containers. In order to increase the sample images and to rectify the class imbalance, data augmentation techniques such as random scaling and flipping were implemented.

Table 1: Development and testing distributions.

Data Type	**Number of samples**
Development	2,363
Testing	587
Total	2,950

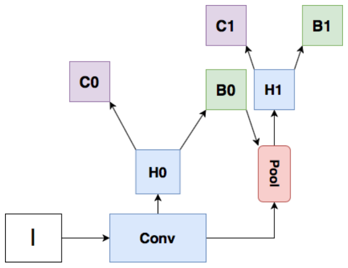
4 Methodology

The two different architectures that was employed are YOLOv4 and Cascade RCNN with Resnest Backbone.

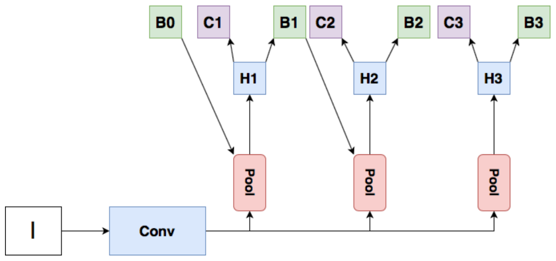
4.1 Cascade RCNN

Cascade RCNN [2] is a multi-stage extension of the two stage architecture of Faster-RCNN. Proposal sub-network is the first stage of the Faster RCNN architecture, in which the entire image is processed by a backbone network after which preliminary detection hypotheses, also called object proposals is produced by applying a proposal head("H0"). The second stage consists of processing the hypotheses by using a region-of-interest detection sub-network ("H1"), denoted as a detection head. Every hypotheses is then assigned a classification score("C") and a bounding box("B"). This architecture also consists of a cascaded bounding box regression and cascaded detection which simultaneously increases both, the quality of hypotheses and the detector.

The Backbone used is ResNeSt over the traditional ResNet.This is because, according to [12], architectures originally designed for image classification like



(a) Faster RCNN



(b) Cascade RCNN

Fig. 1: The architectures of different frameworks. "I" is input image, "conv" backbone convolutions, "pool" region-wise feature extraction, "H" network head, "B" bounding box, and "C" classification. "B0" is proposals in all architectures.

ResNet might not be suitable for downstream applications like object detection because of limited receptive-field size and lack of cross-channel interaction. ResNeSt is made of multiple number of *Split-Attention* blocks stacked in ResNet-style. Each Split-Attention block divides Feature-maps into several groups and finer-grained subgroups or splits, where the feature representation of each group is determined via a weighted combination of the representations of its splits. Feature Pyramid Networks(FPN) is used for feature extraction(referred as pool in Fig 1) while Region Proposal Network(RPN) is used as the proposal head.

4.2 YOLO

Unlike algorithms which re-purposes classifiers for object detection, YOLO takes a different approach. Instead of repeated forward passes, as in the case of sliding window or region proposal models, the image is divided into S*S grids, and a single convolutional network pass outputs the results for all the S*S grids. The output of the network is of the shape S*S*(B * 5 + C), where B is the number of anchor boxes and C is the number of object classes [8]. It generates B number of possible detections for each grid. So instead of separately regressing for region proposals, YOLO considers an end-to-end regression task for detection.

The architecture as mentioned by [1], consists of CSPDarknet53 backbone, SPP additional module, PANet path-aggregation neck, and YOLOv3 head.
YOLOv4 also uses techniques such as 'Bag of Freebies' and 'Bag of Special' to introduce data augmentation, mish activation, cross-stage partial connection and multi-input weighted residual connections in the network .

With YOLOv4, augmentation techniques such as CutMix, Mosaic, class label smoothing and self-adversarial training were trialed to improve the performance. By CutMix, parts of an image is replaced by another image, with annotations from both image parts. Similarly in mosaic augmentation, 4 different images are combined in various ratios, which thereby improves the recognition of objects of different scales. To reduce model overfitting on the classification output, the confidence scores are intentionally reduced by label smoothing technique. Finally by Self-Adversarial training (SAT), the images are passed normally in the forward step. Instead of back propagating, the loss output is used to distort the image in a way it harms the model. These augmented images are then used to train the model in a generic fashion. With these additional improvements YOLOv4 performs more accurately as compared to its earlier versions.

5 Results

In the ImageClef DrawnUI 2020 Challenge, our team CudaMemError1 achieved a top submission rank of 2, with Overall Precision of 0.9504. Two additional metrics namely, mAP@IoU 0.5 and recall@IoU 0.5 were further used to evaluate the models. We can observe that, Cascade RCNN (67972) scored better than

Table 2: Scores on test images

Run ID	**Model**	**Overall precision**	**mAP@IoU 0.5**	**recall@IoU 0.5**
67413	**baseline** Faster RCNN	0.94789	57.2	40.3
67833	Cascade RCNN	0.95035	68.16	53.3
67710	Cascade RCNN	0.94909	64.92	50.5
67722	Cascade RCNN	0.93463	72.33	58.5
67829	YOLOv4	0.93300	73.82	55.6
67707	YOLOv4	0.93125	79.24	59.4
67831	YOLOv4	0.92987	79.11	60
67972	Cascade RCNN	**0.95044**	71.53	55.6
67706	YOLOv4	0.93437	**79.36**	**59.8**

Run 67972: 10000 iterations; Run 67706: 7000 iterations

YOLOv4 (67706) by 1.7% in terms of Overall Precision but, YOLOv4 outperformed Cascade RCNN by 10.9% and 7.5% in mAP@IoU 0.5 and recall@IoU 0.5 respectively. Comparing with the baseline Faster RCNN model, a 38.7% improvement in mAP@IoU 0.5 and 48.3% improvement in recall@IoU 0.5 was achieved with YOLOv4 model and a 25% improvement in mAP@IoU 0.5 and 37.9% in recall@IoU 0.5 is achieved with the Cascade RCNN model.

6 Conclusion

Two different architectures namely YOLOv4 and Cascade RCNN were implemented for detecting UI elements from hand drawn sketches. YOLOv4 performed significantly better than the baseline and Cascade RCNN in terms of mAP@IoU 0.5 and recall@IoU 0.5 with a score of 79.36 and 59.8 respectively, while Cascade RCNN had an Overall Precision score of 0.9504.

Front end development can be streamlined and made easy with these detection systems. Automatic code generation can be further added to this pipeline in order to make the whole process of front end development simple and convenient. These detection systems will be an asset for developers to be able to keep up with ever-growing software engineering world, and also for non-developers to get into the digital space without learning how to code.

References

1. Bochkovskiy, A., Wang, C.Y., Liao, H.Y.M.: Yolov4: Optimal speed and accuracy of object detection (2020)
2. Cai, Z., Vasconcelos, N.: Cascade r-cnn: High quality object detection and instance segmentation (2019)
3. 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)
4. Girshick, R.: Fast r-cnn (2015)
5. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation (2013)
6. He, K., Zhang, X., Ren, S., Sun, J.: Spatial pyramid pooling in deep convolutional networks for visual recognition (2014)
7. 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)
8. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection (2015)
9. Redmon, J., Farhadi, A.: Yolo9000: Better, faster, stronger (2016)
10. Redmon, J., Farhadi, A.: Yolov3: An incremental improvement (2018)
11. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. In: Advances in neural information processing systems. pp. 91–99 (2015)
12. Zhang, H., Wu, C., Zhang, Z., Zhu, Y., Zhang, Z., Lin, H., Sun, Y., He, T., Mueller, J., Manmatha, R., Li, M., Smola, A.: Resnest: Split-attention networks (2020)
13. Zhao, Z.Q., Zheng, P., tao Xu, S., Wu, X.: Object detection with deep learning: A review (2018)