ELEKTRONICA-ICT

Bachelorproef 2022-2023

Object Detection in Automated Software Testing

Author	*Brent Gerets*
Intern	*Bart Stukken*
Extern	*David Vandingenen*
Company	*Brightest*

Abstract

Content

1 Introduction

In recent years, extensive research has resulted in great leaps in the field of machine learning. Many different machine learning models are available to be used by anyone, in a wide variety of applications. One such application that has not received as much attention as others is automated software testing. A reason for this might be that machine learning is a great solution to automate solving problems that previously could only be solved efficiently by humans (e.g. detecting cats in images). Meanwhile, many solutions exist for automated software testing. For example, Selenium is a library available in many languages that facilitates automated testing in the browser [1]. The library allows for interaction with web pages by retrieving the desired elements using XPath, id, class name, etc. [2]. In most cases this works fine. However, being able to retrieve an element using the library is not an assurance that it is visible to the user. Thus, the goal of the project is to discover whether machine learning is a viable solution to this problem. More specifically, object detection is the subcategory of machine learning that applies in this case. The research question can then be formulated as: 'How can object detection be used in the automated software testing process?'.

In more practical terms, the goal of the project is to develop a proof-of-concept of the use of object detection in automated testing. This includes deciding which object detection model(s) to use and developing a user interface to facilitate the training and testing of the object detection models on different web pages, as well as an API that can be used to perform object detection inference in an automated testing environment. Developing an object detection model is not in the scope since there are many pre-trained models available that can be applied to the project. The input of the model consists of screenshots of different pages of the website with bounding boxes around the elements that are to be detected. The model should be able to detect the elements at different positions on the screen and different scales, most other variations should not be detected since this should make the test fail (e.g. an element in the wrong colour should not be detected).

A few papers on the topic of user interface element detection have been published [3, 4, 5, 6, 7, 8]. However, all except one share a common problem that prevents their solutions from being applied to this project. Namely, they attempt to detect as many UI elements as possible and classifying them in certain categories (e.g. text, image, button, slider…). However, the aforementioned use case requires the detection of specific elements to confirm whether they are being displayed and subsequently perform actions on them. For instance, it is not the desired result to classify all buttons on a web page as buttons, but rather to detect all instances of a specific button. *Yeh et al.* [8] did propose a solution for detecting UI elements by providing screenshots of the specific elements. Their solution looks promising but unfortunately the paper was published in 2009, making the technology used in the paper very outdated. Nonetheless their paper can provide some useful insights.

First, section 2 discusses the materials and methods that were employed in the project. More specifically, the programming language and frameworks (2.1), object detection models (2.2), and data augmentation (2.3). Next, section 3 presents the results that were achieved. Section 4 then proceeds to examine and discuss these results. Finally, section 5 formulates a conclusion and reflection.

2 Material and methods

2.1 Programming language and UI framework

Python is used as the programming language for the application, for several reasons. Firstly, Python is the most popular language when it comes to machine learning and data science as a whole, thanks to its ease of use and rich collection of libraries and frameworks [9]. Moreover, Python is and has been the most popular programming language in general in recent years, according to the TIOBE index [10].

NiceGUI [11] is used to create the user interface of the application. It uses the browser as the frontend of the Python code, though the application will exclusively be run locally. The reason for choosing

NiceGUI is its modern look compared to native frameworks such as Tkinter [12]. Also, using NiceGUI would make the possible transition to a website in the future more straightforward.

2.2 Object detection model comparison

To perform object detection, pretrained models are used and fine-tuned on a dataset containing the webpage elements. A large variety of pretrained models is available, so to avoid spending a great deal of time testing all of them, a selection is made based on the most important qualifiers. These were deemed to be accuracy, speed, and availability.

2.2.1 Metrics and models

Firstly, accuracy determines how good a model is at performing object detection on a specific dataset. This dataset is the popular Common Objects in Context (COCO) dataset [13]. Every model that was researched performed a benchmark on the COCO dataset. Most used the COCO validation set, a few used the test-dev set, and some others didn't specify. The accuracies between these different COCO sets are generally very similar. The accuracy on COCO is expressed as the average precision (AP) with an Intersection over Union (IoU) threshold ranging from 0.5 to 0.95 with steps of 0.05 [14]. This threshold determines when a bounding box is accepted as correct. The IoU is calculated as follows [15]:

$$IoU = \frac{Area\ of\ Overlap}{Area\ of\ Union}$$

This means that if the bounding box produced by the model and the true bounding box align perfectly, the IoU will be 1. The AP is usually represented as a percentage and is sometimes also called mean average precision (mAP).

Secondly, speed refers to how much time a model spends to detect objects in an image. This is very dependent on the quality of hardware that is used. Consequently, it is difficult to compare the speed of different models without performing benchmarking for all of them. However, the amount of Floating Point Operations (FLOPs[1]) that the model has to perform on an image can function as an estimate of efficiency and complexity [16]. Therefore, it can also function as an estimate of how much time a model needs to process an image. Some models unfortunately did not mention the number of FLOPs so the Frames Per Second (FPS) was used as the alternative.

Lastly, the availability of a model determines how easy it is to get the model up and running. The availability was split into three categories from low to high availability: download, library, and library without the need for data loading. A model that is only available through download is the hardest to get up and running since the entire setup must be figured out and performed by the developer. A model that is available directly through a library or framework is easier to use because their documentation tends to be relatively thorough. However, the data must still be loaded in and modified appropriately by the developer. Some libraries don't require this and simply use a configuration file, these models are easiest to use and set up. The reason some importance is attached to availability is that high availability ensures the model is functioning correctly and at its best performance.

There is large collection of object detection models available. Comparing every one of them however would be a waste of time. A considerable portion of models are already outdated and easily surpassed by newer state-of-the-art models. The determine which models to compare, three different avenues where explored. First, a brief literature study of object detection model reviews was conducted to ascertain the most popular models. From this literature study can be concluded that YOLO, Faster R-CNN, and SSD are three of the most popular models [17, 18, 19, 20, 21]. Another avenue trough which to find popular models is to check which models are available directly from machine learning frameworks such as PyTorch [22], TensorFlow [23] and Ultralytics [24]. These models are Faster-CNN, SSD, FCOS, RetinaNet, YOLO, RT-DETR, and more. Lastly, another way to find models is to look at the COCO leaderboard [25]. Most of the models that appear in a published paper and have benchmarked on the COCO test-dev set appear on this

[1] Don't confuse FLOPs (Floating Point Operations) with FLOPS (Floating Point Operations per Second) [16].

leaderboard. Currently, Co-DETR is the best performing model in terms of accuracy. Some other state-of-the-art models were found on the leaderboard and compared.

2.2.2 Comparison

Taking inspiration from the work of *Ouchra and Belangour* [26], a Weighted Scoring Model (WSM) was used to determine which models should be the focus of the project. Table 1 contains all the models that were compared together with their accuracy score, speed score, availability score and total weighted score. The scores were derived from their respective specifications. As mentioned, the accuracy was determined using the reported accuracy of the model on the COCO dataset. The accuracy score was then calculated as a value between 0 and 100, where the model with the lowest accuracy has a score of 0, and the highest a score of 100, the rest in between. The speed score was calculated in the same way, lowest amount of FLOPs gets a score of 100, highest a score of 0. If the FLOPs were not reported, the FPS was used as a fallback. As mentioned above, the availability was split into three categories. The lowest availability category receives a score of 0, the middle category 50, and the highest availability category gets a score of 100. Attachment Comparison of object detection models contains the complete table with all the accuracy, FLOPs and FPS metrics, as well as the sources of the data. Finally, a weight between 0 and 1 was assigned to the accuracy speed and availability. This weight represents the expected importance of each metric in the context of the project. Since speed is not imperative, it was assigned a weight of 0.3, while accuracy was assigned a weight of 0.6. The availability receives a very small weight of 0.1 since it is the least important factor that determines the quality of the model. These weights are by no means set in stone, but changing the weights slightly results in the same models receiving the highest score. The weighted total score is then once again a score between 0 and 100. The top 3 models are highlighted in the table below: YOLOv9-E, RT-DETR-X, and Co-DINO. The accuracy of YOLOv9-E and RT-DETR-X is around 10% lower than Co-DINO, which is significant. However, the combination of superior speed and availability places these models right behind Co-DINO. Even if the weights are changed slightly (while still reflecting their importance in the project), the YOLOv9, RT-DETR and Co-DINO models always end up in the top 5. In the case of YOLOv9 and RT-DETR a variant other than the highlighted one could be used since the scores are very similar and the slight increase in accuracy might not be worth the decreased speed. Usually, variants of a model perform similarly. However, Co-DETR is a special case since it is more of a technique that can be applied to existing DETR models. Therefore, the two Co-DETR variants in the table perform very differently due to Co-DINO and Co-Deformable-DETR being completely different DETR models. From this weighted scoring model comparison can be concluded that YOLOv9, RT-DETR, and Co-DINO should be the focus of the project.

Model name	Model variant	Accuracy score	Speed score	Availability score	Weighted total score
YOLOX	s	42.30	98.98	0.00	55.07
	m	57.95	96.33	0.00	63.67
	l	64.79	91.72	0.00	66.39
	x	68.22	84.59	0.00	66.31
YOLOv7		68.46	94.59	0.00	69.45
	X	72.62	89.78	0.00	70.50
	W6	76.77	80.19	0.00	70.12
	E6	79.95	71.44	0.00	69.40
	D6	80.93	54.99	0.00	65.06
	E6E	82.15	52.94	0.00	65.17
YOLOv8	n	34.47	100.00	100.00	60.68
	s	53.06	98.88	100.00	71.50
	m	66.01	96.06	100.00	78.43

	l	72.62	91.17	100.00	80.92
	x	75.06	85.95	100.00	80.82
YOLOv9	S	57.70	98.98	100.00	74.32
	M	68.95	96.16	100.00	80.22
	C	72.86	94.69	100.00	82.12
	E	**79.22**	**89.64**	**100.00**	**84.42**
RT-DETR	L	72.86	94.29	100.00	82.00
	X	**77.26**	**87.29**	**100.00**	**82.55**
Co-DETR	**Co-DINO**	**100.00**	**84.76**	**0.00**	**85.43**
	Co-Deformable-DETR	86.31	51.99	0.00	67.38
Faster R-CNN	R50-FPN	33.74	75.28	50.00	47.83
	R50-FPN	41.56	13.17	50.00	33.89
	R101-FPN	45.97	8.85	50.00	35.23
	X101-FPN	48.41	2.79	50.00	34.88
R-FCN		20.29	0.00	0.00	12.18
SSD	300 VGG16	0.00	34.24	50.00	15.27
FCOS	R50-FPN	40.59	89.80	50.00	56.29
RetinaNet	R50-FPN	32.27	70.77	50.00	45.60
	R50	37.90	11.93	50.00	31.32
	R101	42.05	11.93	50.00	33.81
InternImage	T	63.33	85.26	0.00	63.57
	S	64.79	81.32	0.00	63.27
	B	66.26	72.24	0.00	61.43
	L	80.44	21.60	0.00	54.74
	XL	80.68	0.00	0.00	48.41
Weight		**0.60**	**0.30**	**0.10**	

Table 1: Weighted Score Model comparison of popular and/or state-of-the-art object detection models

2.3 Data augmentation

Data augmentation is the practice of expanding a dataset by performing certain transformations on the elements of the dataset. This allows the model to learn invariant features and prevents overfitting [27]. It also increases the size of the dataset, which is important in this project since the dataset only consists of a few screenshots of the website. There are numerous augmentations that can be performed such as resizing, rotating, translating, changing colour, flipping, adding noise, etc. [28]. However, a lot of the augmentations are not desired in this project. For example, changing the colour of the images in the dataset is not desired since the model should only detect the elements in the correct colour. The same goes for flipping, rotating, and shearing. There are a few data augmentations that can be applied in the context of this project. Firstly, the images can be resized to allow the model to detect the elements at different sizes. This is desired since changing the size of the browser window could change the dimensions of the elements within. It is important to resize only the part of the image within the bounding box. Resizing the entire image would be pointless since the image must be resized again before being input into the model. Secondly, the part of the image withing the bounding box can be translated to a different position inside the image to prevent the model from overfitting to a specific position. Lastly, even though the model shouldn't be invariant to colour, noise can still be added to prevent overfitting and to allow the model to still detect the element when there are other elements in the vicinity. These are the most important data augmentations that can be applied. It should be noted that these augmentations should be applied to each other as well. For example, the original images are randomly resized. Then all the images, including the resized ones, are randomly translated. Finally, noise is added to all the images. Depending on the number of times an augmentation is applied, this can exponentially increase the size of the dataset.

3 Results

4 Discussion

5 Conclusion

6 Reference list

[1] Selenium, „Getting started," [Online]. Available: https://www.selenium.dev/documentation/webdriver/getting_started/ [Accessed: 02/04/2024].

[2] Selenium, „Locators," [Online]. Available: https://www.selenium.dev/documentation/webdriver/elements/locators/ [Accessed: 02/04/2024].

[3] J. Chen, M. Xie, Z. Xing, C. Chen, X. Xu, L. Zhu en G. Li, „Object detection for graphical user interface: old fashioned or deep learning or a combination?," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020.

[4] A. Kumar, K. Morabia, W. Wang, K. Chang en A. Schwing, „CoVA: Context-aware Visual Attention for Webpage Information Extraction," in *Proceedings of The Fifth Workshop on e-Commerce and NLP (ECNLP 5)*, 2022.

[5] T. Gogar, O. Hubacek en J. Sedivy, „Deep Neural Networks for Web Page Information Extraction," in *Artificial Intelligence Applications and Innovations*, Cham, 2016.

[6] M. Altinbas en T. Serif, „GUI Element Detection from Mobile UI Images Using YOLOv5," 2022, pp. 32-45.

[7] M. Xie, „UIED," [Online]. Available: https://github.com/MulongXie/UIED?tab=readme-ov-file [Accessed: 27/03/2024].

[8] T. Yeh, T.-H. Chang en R. C. Miller, „Sikuli: using GUI screenshots for search and automation," in *Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology*, New York, NY, USA, 2009.

[9] J. C. Luna, „Top programming languages for data scientists in 2023," [Online]. Available: https://www.datacamp.com/blog/top-programming-languages-for-data-scientists-in-2022 [Accessed: 02/04/2024].

[10] TIOBE, „TIOBE Index," [Online]. Available: https://www.tiobe.com/tiobe-index/ [Accessed: 02/04/2024].

[11] NiceGUI, „NiceGUI," [Online]. Available: https://nicegui.io/ [Accessed: 02/04/2024].

[12] M. Roseman, „TkDocs," [Online]. Available: https://tkdocs.com/index.html [Accessed: 02/04/2024].

[13] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár en C. L. Zitnick, „Microsoft COCO: Common Objects in Context," in *Computer Vision – ECCV 2014*, Cham, 2014.

[14] COCO, „Detection evaluation," [Online]. Available: https://cocodataset.org/#detection-eval [Accessed: 13/03/2024].

[15] D. Shah, „Intersection over Union (IoU): Definition, Calculation, Code," [Online]. Available: https://www.v7labs.com/blog/intersection-over-union-guide [Accessed: 13/03/2024].

[16] D. Li, „Calculate Computational Efficiency of Deep Learning Models with FLOPs and MACs," [Online]. Available: https://www.kdnuggets.com/2023/06/calculate-computational-efficiency-deep-learning-models-flops-macs.html [Accessed: 13/03/2024].

[17] O. E. Olorunshola, A. K. Ademuwagun en C. Dyaji, „Comparative Study of Some Deep Learning Object Detection Algorithms: R-CNN, FAST RCNN, FASTER R-CNN, SSD, and YOLO," *Nile Journal of Engineering & Applied Science,* vol. 1, p. 70–80, September 2023.

[18] S. A. Sanchez, H. J. Romero en A. D. Morales, „A review: Comparison of performance metrics of pretrained models for object detection using the TensorFlow framework," *IOP Conference Series: Materials Science and Engineering,* vol. 844, p. 012024, May 2020.

[19] S. Srivastava, A. V. Divekar, C. Anilkumar, I. Naik, V. Kulkarni en V. Pattabiraman, „Comparative analysis of deep learning image detection algorithms," *Journal of Big Data,* vol. 8, p. 66, 2021.

[20] Z.-Q. Zhao, P. Zheng, S.-T. Xu en X. Wu, „Object Detection With Deep Learning: A Review," *IEEE Transactions on Neural Networks and Learning Systems,* vol. PP, pp. 1-21, January 2019.

[21] H. Zota en M. Dhande, „A Comparative Study of Widely Used Image Detection Algorithms," *International Research Journal of Engineering and Technology,* vol. 7, p. 5183–5187, July 2020.

[22] PyTorch, „Models and pre-trained weights: Object detection," [Online]. Available: https://pytorch.org/vision/stable/models.html#object-detection [Accessed: 13/03/2024].

[23] Kaggle, „Models," [Online]. Available: https://www.kaggle.com/models?tfhub-redirect=true&task=17074&fine-tunable=true [Accessed: 13/03/2024].

[24] G. Jocher en Laughing-q, „Models Supported by Ultralytics," 12 November 2023. [Online].

[25] PapersWithCode, „Object Detection on COCO test-dev," [Online]. Available: https://paperswithcode.com/sota/object-detection-on-coco [Accessed: 13/03/2024].

[26] H. Ouchra en A. Belangour, „Object Detection Approaches in Images: A Weighted Scoring Model based Comparative Study," *International Journal of Advanced Computer Science and Applications,* vol. 12, 2021.

[27] B. Zoph, E. D. Cubuk, G. Ghiasi, T.-Y. Lin, J. Shlens en Q. V. Le, „Learning Data Augmentation Strategies for Object Detection," *CoRR,* vol. abs/1906.11172, 2019.

[28] J. Nelson, „What is Image Preprocessing and Augmentation?," 26 January 2020. [Online]. Available: https://blog.roboflow.com/why-preprocess-augment/ [Accessed: 20/03/2024].

[29] Z. Ge, S. Liu, F. Wang, Z. Li en J. Sun, „YOLOX," [Online]. Available: https://github.com/Megvii-BaseDetection/YOLOX [Accessed: 13/03/2024].

[30] C.-Y. Wang, A. Bochkovskiy en H.-Y. M. Liao, „yolov7," [Online]. Available: https://github.com/WongKinYiu/yolov7 [Accessed: 13/03/2024].

[31] G. Jocher, A. Chaurasia, F. C. Akyon en Laughing-q, „YOLOv8," 12 November 2023. [Online]. Available: https://docs.ultralytics.com/models/yolov8/ [Accessed: 13/03/2024].

[32] G. Jocher, B. Qaddoumi en Laughing-q, „YOLOv9: A Leap Forward in Object Detection Technology," 26 February 2024. [Online]. Available: https://docs.ultralytics.com/models/yolov9/ [Accessed: 13/03/2024].

[33] G. Jocher, „Baidu's RT-DETR: A Vision Transformer-Based Real-Time Object Detector," 12 November 2023. [Online]. Available: https://docs.ultralytics.com/models/rtdetr/ [Accessed: 13/03/2024].

[34] Z. Zong, G. Song en Y. Liu, „Co-DETR," [Online]. Available: https://github.com/Sense-X/Co-DETR [Accessed: 13/03/2024].

[35] PyTorch, „Faster R-CNN," [Online]. Available: https://pytorch.org/vision/main/models/faster_rcnn.html [Accessed: 13/03/2024].

[36] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo en R. Girshick, *Detectron2,* 2019.

[37] J. Dai, Y. Li, Y. Xiong en H. Qi, „R-FCN," [Online]. Available: https://github.com/daijifeng001/R-FCN [Accessed: 13/03/2024].

[38] PyTorch, „SSD," [Online]. Available: https://pytorch.org/vision/main/models/ssd.html [Accessed: 13/03/2024].

[39] PyTorch, „FCOS," [Online]. Available: https://pytorch.org/vision/main/models/fcos.html [Accessed: 13/03/2024].

[40] PyTorch, „RetinaNet," [Online]. Available: https://pytorch.org/vision/main/models/retinanet.html [Accessed: 13/03/2024].

[41] W. Wang, J. Dai, Z. Chen, Z. Huang, Z. Li, X. Zhu, X. Hu, T. Lu, L. Lu, H. Li en others, „InternImage," [Online]. Available: https://github.com/opengvlab/internimage [Accessed: 13/03/2024].

[42] PapersWithCode, „Torchvision," [Online]. Available: https://paperswithcode.com/lib/torchvision [Accessed: 13/03/2024].

[43] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu en A. C. Berg, „SSD: Single Shot MultiBox Detector," in *Computer Vision – ECCV 2016*, Cham, 2016.

[44] PapersWithCode, „NAS-FCOS: Fast Neural Architecture Search for Object Detection," [Online]. Available: https://paperswithcode.com/paper/nas-fcos-fast-neural-architecture-search-for [Accessed: 13/03/2024].

7 Attachment

7.1 Comparison of object detection models

Model		Accuracy		
Name	**Variant**	**Score**	**mAP (%)**	**Dataset**
YOLOX	s	42.30	40.50	COCO
	m	57.95	46.90	COCO
	l	64.79	49.70	COCO
	x	68.22	51.10	COCO
YOLOv7		68.46	51.20	COCO val
	X	72.62	52.90	COCO val
	W6	76.77	54.60	COCO val
	E6	79.95	55.90	COCO val
	D6	80.93	56.30	COCO val
	E6E	82.15	56.80	COCO val
YOLOv8	n	34.47	37.30	COCO val
	s	53.06	44.90	COCO val
	m	66.01	50.20	COCO val
	l	72.62	52.90	COCO val
	x	75.06	53.90	COCO val
YOLOv9	S	57.70	46.80	COCO val
	M	68.95	51.40	COCO val
	C	72.86	53.00	COCO val
	E	**79.22**	**55.60**	**COCO val**
RT-DETR	L	72.86	53.00	COCO val
	X	**77.26**	**54.80**	**COCO val**
Co-DETR	**Co-DINO**	**100.00**	**64.10**	**COCO val**
	Co-Deformable-DETR	86.31	58.50	COCO val
Faster R-CNN	R50-FPN	33.74	37.00	COCO val
	R50-FPN	41.56	40.20	COCO val
	R101-FPN	45.97	42.00	COCO val
	X101-FPN	48.41	43.00	COCO val
R-FCN		20.29	31.50	COCO val
SSD	300 VGG16	0.00	23.20	COCO test-dev
FCOS	R50-FPN	40.59	39.80	COCO test-dev
RetinaNet	R50-FPN	32.27	36.40	COCO minival
	R50	37.90	38.70	COCO val
	R101	42.05	40.40	COCO val
InternImage	T	63.33	49.10	COCO
	S	64.79	49.70	COCO
	B	66.26	50.30	COCO
	L	80.44	56.10	COCO
	XL	80.68	56.20	COCO
Weight		**0.60**		

Model		Speed		
Name	**Variant**	**Score**	**FLOPs (B)**	**FPS**
YOLOX	s	98.98	26.80	102.04
	m	96.33	73.80	81.30
	l	91.72	155.60	68.97
	x	84.59	281.90	57.80
YOLOv7		94.59	104.70	161.00
	X	89.78	189.90	114.00
	W6	80.19	360.00	84.00
	E6	71.44	515.20	56.00
	D6	54.99	806.80	44.00
	E6E	52.94	843.20	36.00
YOLOv8	n	100.00	8.70	
	s	98.88	28.60	
	m	96.06	78.60	
	l	91.17	165.20	
	x	85.95	257.80	
YOLOv9	S	98.98	26.70	
	M	96.16	76.80	
	C	94.69	102.80	
	E	**89.64**	**192.50**	
RT-DETR	L	94.29	110.00	
	X	**87.29**	**234.00**	
Co-DETR	**Co-DINO**	**84.76**	**279.00**	
	Co-Deformable-DETR	51.99	860.00	
Faster R-CNN	R50-FPN	75.28	447.00	10.20
	R50-FPN	13.17		26.32
	R101-FPN	8.85		19.61
	X101-FPN	2.79		10.20
R-FCN		0.00		5.88
SSD	300 VGG16	34.24		59.00
FCOS	R50-FPN	89.80	189.60	
RetinaNet	R50-FPN	70.77	527.00	24.39
	R50	11.93		24.39
	R101	11.93		24.39
InternImage	T	85.26	270.00	
	S	81.32	340.00	
	B	72.24	501.00	
	L	21.60	1399.00	
	XL	0.00	1782.00	
Weight		**0.30**		

Model		Availability		
Name	**Variant**	**Score**	**How**	**Where**
YOLOX	s	0.00	download	[29]
	m	0.00	download	

	l	0.00	download	
	x	0.00	download	
YOLOv7		0.00	download	[30]
	X	0.00	download	
	W6	0.00	download	
	E6	0.00	download	
	D6	0.00	download	
	E6E	0.00	download	
YOLOv8	n	100.00	lib, no data loading	[31]
	s	100.00	lib, no data loading	
	m	100.00	lib, no data loading	
	l	100.00	lib, no data loading	
	x	100.00	lib, no data loading	
YOLOv9	S	100.00	lib, no data loading	[32]
	M	100.00	lib, no data loading	
	C	100.00	lib, no data loading	
	E	**100.00**	**lib, no data loading**	
RT-DETR	L	100.00	lib, no data loading	[33]
	X	**100.00**	**lib, no data loading**	
Co-DETR	**Co-DINO**	**0.00**	**download**	[34]
	Co-Deformable-DETR	0.00	download	
Faster R-CNN	R50-FPN	50.00	lib	[35]
	R50-FPN	50.00	lib	[36]
	R101-FPN	50.00	lib	
	X101-FPN	50.00	lib	
R-FCN		0.00	download	[37]
SSD	300 VGG16	50.00	lib	[38]
FCOS	R50-FPN	50.00	lib	[39]
RetinaNet	R50-FPN	50.00	lib	[40]
	R50	50.00	lib	[36]
	R101	50.00	lib	
InternImage	T	0.00	download	[41]
	S	0.00	download	
	B	0.00	download	
	L	0.00	download	
	XL	0.00	download	
Weight		**0.10**		

Model		Scoring		Data source
Name	**Variant**	**Weighted score**	**Rank**	
YOLOX	s	55.07	27	[29]
	m	63.67	21	
	l	66.39	17	
	x	66.31	18	
YOLOv7		69.45	14	[30]
	X	70.50	12	

	W6	70.12	13	
	E6	69.40	15	
	D6	65.06	20	
	E6E	65.17	19	
YOLOv8	n	60.68	25	[31]
	s	71.50	11	
	m	78.43	9	
	l	80.92	6	
	x	80.82	7	
YOLOv9	S	74.32	10	[32]
	M	80.22	8	
	C	82.12	4	
	E	**84.42**	**2**	
RT-DETR	L	82.00	5	[33]
	X	**82.55**	**3**	
Co-DETR	**Co-DINO**	**85.43**	**1**	[34]
	Co-Deformable-DETR	67.38	16	
Faster R-CNN	R50-FPN	47.83	30	[42]
	R50-FPN	33.89	34	[36]
	R101-FPN	35.23	32	
	X101-FPN	34.88	33	
R-FCN		12.18	38	[37]
SSD	300 VGG16	15.27	37	[43]
FCOS	R50-FPN	56.29	26	[44]
RetinaNet	R50-FPN	45.60	31	[42]
	R50	31.32	36	[36]
	R101	33.81	35	
InternImage	T	63.57	22	[41]
	S	63.27	23	
	B	61.43	24	
	L	54.74	28	
	XL	48.41	29	
Weight				