TRAINING DETECTRON2 OBJECT DETECTION MODEL TO CLASSIFY VEHICLES

Tushar Kant Samantaray, Rishi Raj

Indiana University Bloomington

Report

ENGR-E 503 Intelligent Systems Engineering

Abstract

Object detection is a well-studied field in Artificial Intelligence that allows robots and other AI technologies to recognize and identify distinct things and entities from images and movies. Detectron2, a state-of-the-art toolset for object detection and segmentation designed by Facebook AI's research team and built on top of PyTorch, has made significant progress in the field. The detectron2 library's simplicity and encapsulation make it a good approach to design and build object detection models without getting too deep into the underlying architecture. Further, we demonstrate how pre-trained models could be used to infer on new data for various object-detection and segmentation tasks.

Keywords: Object-Detection, Detectron2, Vehicle Detection, Colab, PyTorch

TRAINING DETECTRON2 OBJECT DETECTION MODEL TO CLASSIFY VEHICLES

Facebook AI Research (FAIR) introduced the Detectron2 package [2], which produces excellent performance on object detection and segmentation issues. Detectron2 is written in PyTorch and is based on the maskrcnn benchmark. It requires CUDA because the computing costs are rather high.

Our project begins with the creation of a Machine Learning model that can distinguish between five different vehicle classes: automobile, bus, motorcycle, ambulance, and truck. We used an open-source dataset accessible on the Roboflow website [8] to train a Faster R-CNN model on our dataset over a baseline model given by detectron2 and trained by experimenting with various sets of hyperparameters to produce a more accurate model. There are approximately 830 photos in our dataset, which are divided into test, validation, and train subsets. The photos have a resolution of 1024x751 pixels, and the car appears to be overrepresented in the sample. COCO-Dataset format is required by Detectron2, which is essentially a JSON file holding metadata about the photos, such as labels, classes, dimensions, and so on.

The experiment's second portion shows how multiple detectron2 pre-trained models may be used to conduct direct inference on new data for tasks including image segmentation, key-point identification, LVIS segmentation, Panoptic segmentation.

I. Overview of Faster R-CNN model

Before we can train our data, we need to import a baseline model from Detectron2 and set up the model's configurations. For our training, we use a Faster R-CNN model. A group of Microsoft researchers developed the Faster R-CNN model [9], which is a deep convolutional network used for object detection that appears to users as a single end-to-end unified network. Faster R-CNN constructs a region-proposal network that can generate region proposals, which are then input into the detection model (Fast R-CNN) for object inspection [13].

The general architecture of Faster R-CNN is shown below, and it consists of 2 modules:

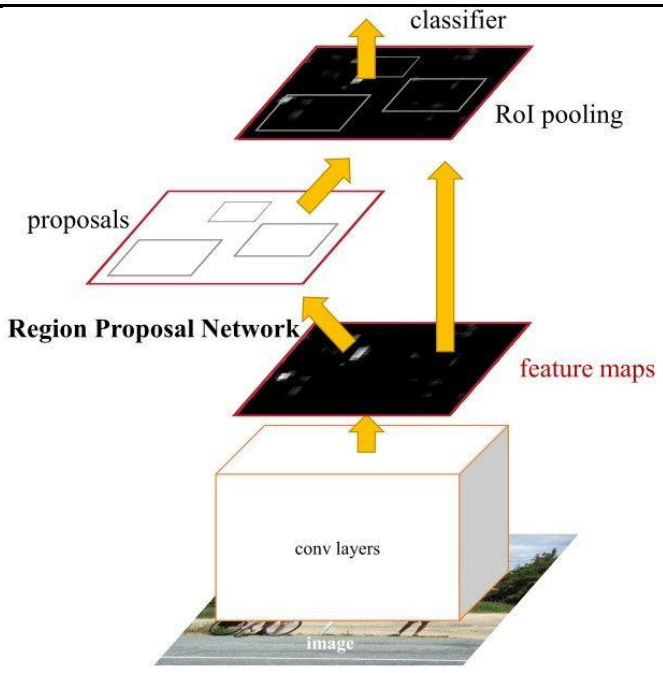

Fig. 1: General architecture of Faster R-CNN

1. RPN: For generating region proposals, the Fast R-CNN detection module is guided to where to seek for items in the image by using the idea of attention in a neural network.
2. R-CNN Fast: For finding items in the suggested regions. The ROI pooling layer extracts a fixed-length feature vector from each region once the region proposal is formed for all region proposals in the pictures. Fast R-CNN is used to classify the retrieved features, and the class scores of the discovered objects, as well as their bounding boxes, are returned.

II. Experimental Setup

We use Google Colab to do our coding and train our model. The following set of libraries, with their following specific versions, are required to support Detectron2 [5]:

1. Torch v1.5.0
2. TorchVision v0.5
3. Pyyaml v5.1
4. Cocoapi
5. Pycocotools-2.0
6. Numpy
7. Matplotlib and OpenCV libraries for visualizing

The following code could be executed directly on Colab to download and install the libraries.

```
!pip install -U torch==1.5 torchvision==0.6 -f
https://download.pytorch.org/whl/cu101/torch_stable.html
!pip install cython pyyaml==5.1
!pip install -U 'git+https://github.com/cocodataset/cocoapi.git#subdirectory=PythonAPI'
```

Once the libraries are downloaded, we need to set the hardware accelerator to ‘GPU’, and then install a version of Detectron2 library that is stable with the current Torch and CUDA version.

8. Detectron2 v0.1.3

```
!pip install detectron2==0.1.3 -f
https://dl.fbaipublicfiles.com/detectron2/wheels/cu101/torch1.5/index.html
```

III. Dataset

The photos must be in 'COCO format' for Detectron2 to work. The "COCO format" is a JSON structure that contains information on how labels and metadata for an image dataset are saved. We get our data from the open-source Roboflow public datasets [8], which allow users to add their tailored data. We use the following commands to download and extract the data directly in Colab:

```
!curl -L "https://public.roboflow.com/ds/20BnsvX8UE?key=HbdJ3kaKcQ" > roboflow.zip
!unzip roboflow.zip
!rm roboflow.zip
```

A. Sample output:

Our dataset has 830 images, grouped into test, validation, and train subsets. The images are 1024x751 pixels, and the car is overrepresented in the dataset. The training dataset contains 454 images, the validation dataset contains 250 images, and the test dataset has 126 images.

Once the training dataset is registered, we could look at the training data. We view three random images from our dataset with their annotations as follows:

```
for d in random.sample(dataset_dicts, 3):
    img = cv2.imread(d["file_name"])
    visualizer = Visualizer(img[:, :, ::-1], metadata=my_dataset_train_metadata,
scale=0.5)
    vis = visualizer.draw_dataset_dict(d)
    plt.figure(figsize=(16, 9))
    plt.imshow(vis.get_image()[:,:,::-1])
```

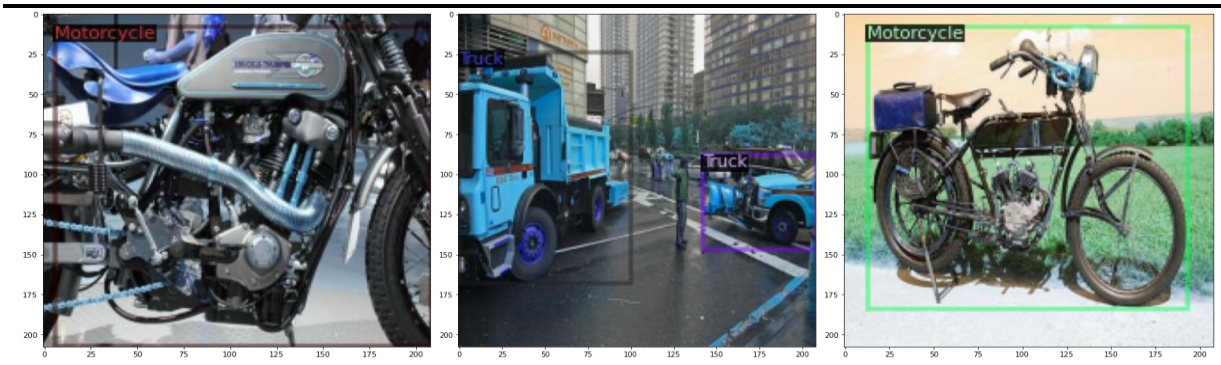

Fig. 2: Viewing random samples of the dataset with annotations

IV. Method

Once we have our data prepared, we need to register the training, validation, and testing datasets. We register our datasets using *DatasetCatalog.register* and the *MetadataCatalog* methods as follows:

```
register_coco_instances("vehicle_dataset_train", {}, "train/_annotations.coco.json", "train")
register_coco_instances("vehicle_dataset_val", {}, "valid/_annotations.coco.json", "valid")
register_coco_instances("vehicle_dataset_test", {}, "test/_annotations.coco.json", "test")

# Create metadata catalog and dataset catalog.
my_dataset_train_metadata = MetadataCatalog.get("vehicle_dataset_train")

dataset_dicts = DatasetCatalog.get("vehicle_dataset_train")
```

A. Training and Evaluation

We used a Faster R-CNN baseline model and its weights configuration with 32 groups each having a width of 8 and a RESNET depth of 101, without a mask (faster_rcnn_X_101_32x8d_FPN_3x) for training. For initial training, we used the following set of hyperparameters:

```
NUM_WORKERS = 4, IMS_PER_BATCH = 4, BASE_LR = 0.01, WARM_UP_ITERS = 50, MAX_ITER = 200,
STEPS = (500, 1500), GAMMA = 0.05, BATCH_SIZE_PER_IMAGE = 128, NUM_CLASSES = 5,
EVAL_PERIOD = 500
```

The training took about 18 minutes, and the evaluation result for bbox were as follow:

TABLE 1: Per-category bbox AP for initial model:

category	AP	category	AP	category	AP
vehicles	nan	Ambulance	51.377	Bus	31.983
Car	23.977	Motorcycle	37.301	Truck	0.000

A score of 40 or higher on the AP scale is generally regarded good. Our first model, on the other hand, isn't performing well in almost all the classes. Although the existing model could categorize vehicles and motorbikes in general, it was unable to recognize ambulances and trucks, misclassifying them as cars.

B. Results

Following are few inferencing test results on the model:

Fig. 3: Inference made on test data using the initial trained model

V. Improving the model

We experimented with a variety of hyperparameters to improve our model, and the following set of hyperparameters was employed in our final model.

```
NUM_WORKERS = 4, IMS_PER_BATCH = 4, BASE_LR = 0.005, WARM_UP_ITERS = 1000, MAX_ITER = 1500,
STEPS = (500, 1500), GAMMA = 0.05, BATCH_SIZE_PER_IMAGE = 128, NUM_CLASSES = 5,
EVAL_PERIOD = 500
```

A. Training and Evaluation

The model takes about 2 hours 15 minutes to train, and below are its evaluation scores:

TABLE 2: Per-category bbox AP for improved model:

category	AP	category	AP	category	AP
vehicles	nan	Ambulance	66.653	Bus	59.287
Car	40.642	Motorcycle	37.076	Truck	0.000

B. Results

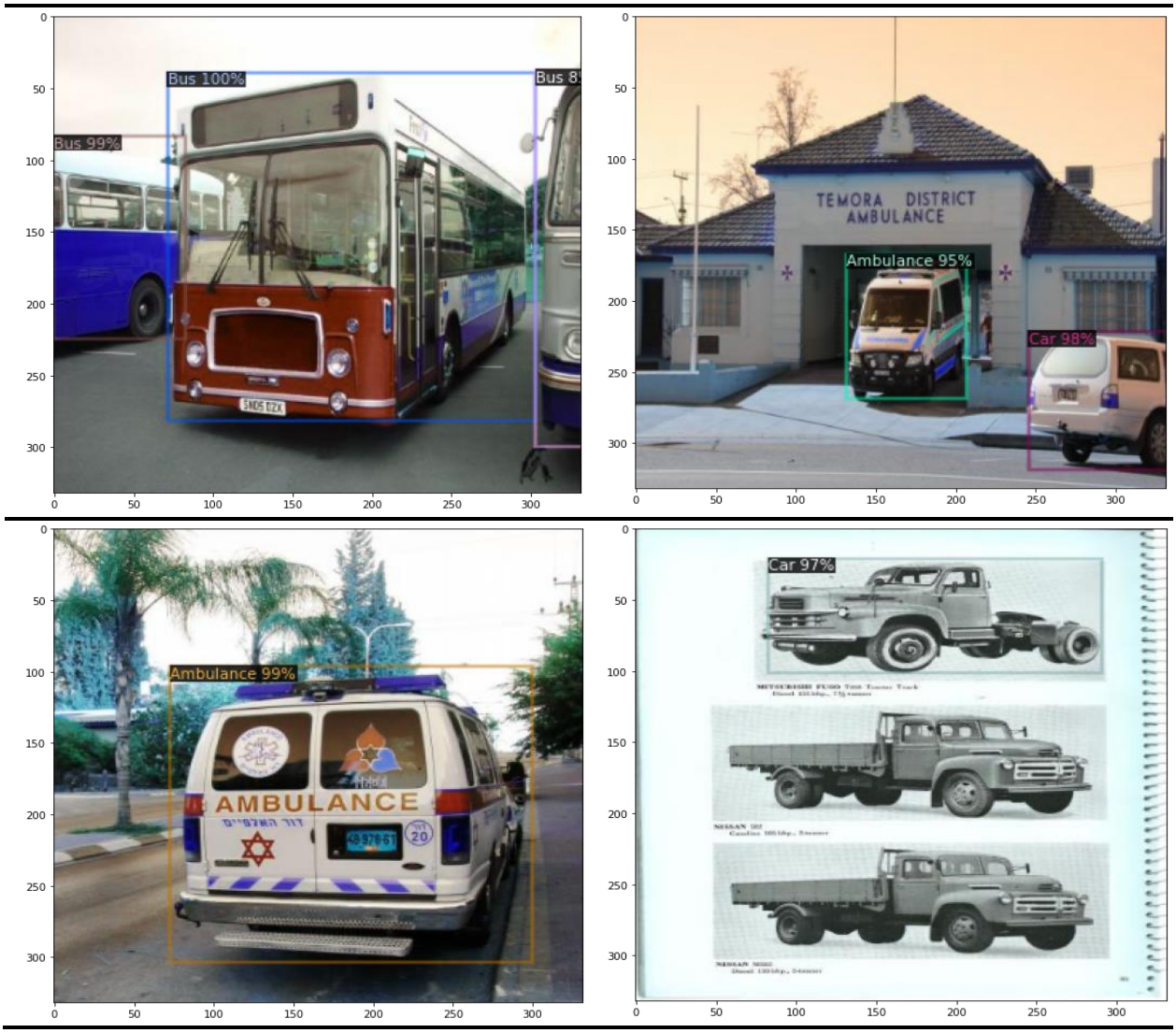

Fig. 4: Inference made on test data using the improved trained model

Except for trucks, the AP scores of each category have climbed. The inferences illustrated in Fig. 4 are instances of this model's inferences.

The enhanced model, in comparison to the previous model, can classify ambulances. It is still unable to categorize the trucks, presumably because of the training dataset's underrepresentation of vehicles. Because cars are overrepresented in the training sample, both models have no trouble identifying them.

VI. Inferencing on pre-trained model

Besides object detection, Detecton2 provides tools to build instance-segmentation, key-point detection too. To demonstrate these features, we import a pre-trained model and do inference using test images.

A. Instance-Segmentation

Detecting and delineating distinct objects of interest in images and videos is called instance segmentation. It segments different instances of each semantic category and thus appears as an extension of semantic segmentation (It identifies different classes of object in an image such as person, book, etc.). In our case, semantic segmentation identifies all the vehicles and then instance segmentation further identifies the individual vehicle type. The metric used for evaluation is average precision. It requires assignment of confidence score to each segment for estimation of precision curve [14].

Model – Faster R-CNN FPN baseline model and its pre-trained weights with 32 groups each having a width of 8 and a RESNET depth of 101, without a mask (faster_rcnn_R_101_FPN_3x), for inferencing.

Result

Fig. 5: Instance-Segmentation Result

The pre-trained model is successfully to segment out all buses in the images.

B. Key-points Detection

Key-points detection is locating key object parts in an image or video. It involves synchronously detecting people and localizing their key points. Key points are spatial location within the image that defines what stands out in the image such as the facial landmarks, the body-joints in a person the corners and blobs in an image. This method is invariant to image rotation, translation, distortion, etc. [12].

Model – Key-points R-CNN FPN baseline model and its pre-trained weights with RESNET depth of 50 (keypoint_rcnn_R_50_FPN_3x), for inferencing.

Result

Fig. 6: Key-points Detection Result

The model used has been trained to recognize people in images and can detect everyone in the image.

C. LVIS Instance-Segmentation

LVIS is a large vocabulary instance segmentation system that can segment for over 1000 object categories at the entry level. It contains over 2.2 million high quality instance segmentation masks for over 1000 entry level object categories in 164K image [7]. Object categories in LVIS are divided into rare, common, frequent sets respectively containing <10, 10-100 and >=100 training images. We use Average Precision as evaluation metric.

Model – Mask R-CNN FPN baseline model and its pre-trained weights with RESNET depth of 101 (mask_rcnn_R_101_FPN_1x), for inferencing.

Result

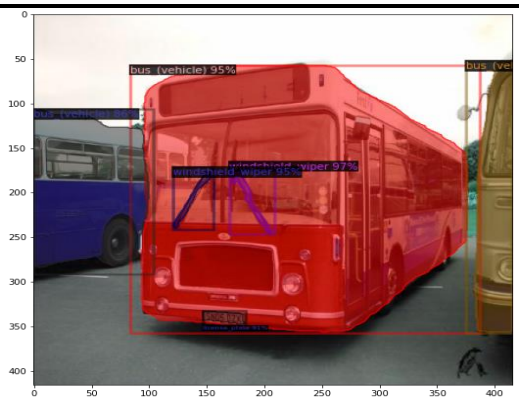

Fig. 7: LVIS Instance-Segmentation Result

LVIS Instance segmentation is effective in identifying the image's most detailed object. It can recognize the bus's windshield and wiper in the case presented in fig. 7, which was not achievable with Panoptic segmentation.

D. Panoptic-Segmentation

The task of distinguishing distinct classes of objects in an image is known as panoptic segmentation. It is a combination of semantic segmentation and instance segmentation. In other words, it semantically distinguishes different object as well as identifies separate instance of each kind of object in an input image. It divides the image in two parts “things” and “stuffs”. Here, things are the objects that are countable in Computer Vision projects such as person, vehicle, cat, etc. And stuff is the uncountable amorphous region of identical texture such as sky and road in our case. The metric used is panoptic quality (PQ) [10].

PQ = Average IoU of matched segments * F1-score

Model – Panoptic R-CNN FPN baseline model and its pre-trained weights with RESNET depth of 101 (panoptic_fpn_R_101_3x), for inferencing.

Result

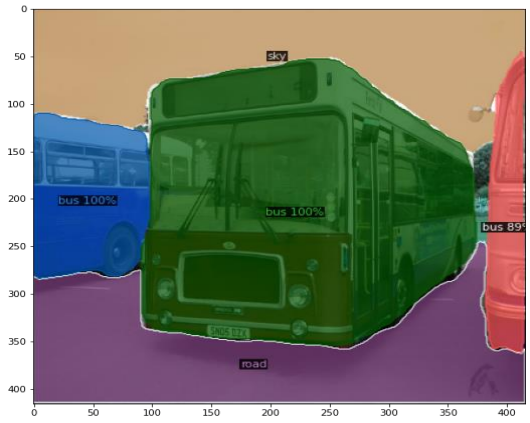

Fig. 8: Panoptic-Segmentation Result

The result of a Panoptic Segmentation is shown in Fig. 8. Aside from buses, it can also divide the road and the sky.

Conclusions

Detectron2 is a cutting-edge library that allows you to create object detection and segmentation models. Its encapsulation makes it a popular choice for creating object-detection models. It includes capabilities for instance segmentation, key-point detection, panoptic, and LVIS-instance segmentation in addition to object detection. In comparison to other frameworks like YOLO, which have an obscure format of their scoring results delivered in multidimensional array objects, Detectron2 is a high-performance and simpler to use upgrade of Detectron2 and originating from Mask R-CNN benchmark providing easy API to extract soring results.

The model is well trained when a low learning rate is combined with many training epochs. A training class's underrepresentation leads to poor inference for that class. Because it deals with deep network models on enormous image data, the computing cost is still considerable. Detectron2 requires a CUDA device to run in a Windows environment because it is not well maintained. Although Google Colab provides a good option for training the detectron2 model, the runtime time constraint is a problem that stopped us from trying more hyperparameter configurations.

More work may be done to improve the model, which is still unable to adequately detect trucks and perform object detection on movies. This project also lays the groundwork for future projects that will create and use our own bespoke datasets to address specific objects to detect.

Contributions

1. Tushar Kant Samantaray – Environment Setup, Coding, Training, Presentation, Report Writing.
2. Rishi Raj – Model Evaluation, Presentation, Report Writing.

References

[1]" C. Zhang, "How to train Detectron2 with Custom COCO Datasets", *Medium*, 2021. [Online]. Available: https://medium.com/@chengweizhang2012/how-to-train-detectron2-with-custom coco-datasets-4d5170c9f389. [Accessed: 12- Dec- 2021]

[2]" GitHub - facebookresearch/detectron2: Detectron2 is a platform for object detection, segmentation and other visual recognition tasks.", *GitHub*, 2021. [Online]. Available: https://github.com/facebookresearch/detectron2.

[3]" Use Custom Datasets — detectron2 0.6 documentation", *Detectron2.readthedocs.io*, 2021. [Online]. Available: https://detectron2.readthedocs.io/en/latest/tutorials/datasets.html.

[4]" DETECTRON2 – 4 Advanced Computer Vision tasks using the pre-trained Detectron2 – Bugspeed", *Bugspeed.xyz*, 2021. [Online]. Available: https://bugspeed.xyz/detectron2-4-advanced-computer-vision-tasks-using-the-pre-trained-detectron2/. [Accessed: 12- Dec- 2021].

[5]" J. Solawetz, "How to Train Detectron2 on Custom Object Detection Data", *Roboflow Blog*, 2021. [Online]. Available: https://blog.roboflow.com/how-to-train-detectron2/. [Accessed: 12- Dec- 2021].

[6]" R. Shri Varsheni, "Your Guide to Object Detection with Detectron2 in PyTorch", *Analytics Vidhya*, 2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/08/your-guide-to-object-detection-with-detectron2-in-pytorch/. [Accessed: 12- Dec- 2021].

[7]" A. Gupta, P. Doll´ar and R. Girshick, *Openaccess.thecvf.com*, 2021. [Online]. Available: https://openaccess.thecvf.com/content_CVPR_2019/papers/Gupta_LVIS_A_Dataset_for_Large_Vocabulary_Instance_Segmentation_CVPR_2019_paper.pdf.

[8]" J. Solawetz, "Vehicles-OpenImages Object Detection Dataset", *Roboflow*, 2021. [Online]. Available: https://public.roboflow.com/object-detection/vehicles-openimages. [Accessed: 12- Dec- 2021].

[9]" A. KHAZRI, "Faster RCNN Object detection", *Medium*, 2021. [Online]. Available: https://towardsdatascience.com/faster-rcnn-object-detection-f865e5ed7fc4. [Accessed: 12- Dec- 2021].

[10]" A. Kirillov, P. Doll´ar, C. Rother, R. Girshick and K. He, *Openaccess.thecvf.com*, 2021. [Online]. Available: https://openaccess.thecvf.com/content_CVPR_2019/papers/Kirillov_Panoptic_Segmentation_CVPR_2019_paper.pdf. [Accessed: 12- Dec- 2021].

[11]" J. Adamczyk, "Understanding Detectron2 demo", *Medium*, 2021. [Online]. Available: https://towardsdatascience.com/understanding-detectron2-demo-bc648ea569e5. [Accessed: 12- Dec- 2021].

[12]" Zhang, J., Chen, Z., & Tao, D. (2021). Towards high performance human keypoint detection. *International Journal of Computer Vision*, *129*(9), 2639-2662.

[13]" R. Gandhi, " R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms", *Medium*, 2021. [Online]. Available: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e. [Accessed: 12- Dec- 2021].

[14" Hafiz, A. M., & Bhat, G. M. (2020). A survey on instance segmentation: state of the art. *international journal of multimedia information retrieval*, 1-19.