Robot Grasp Detection Using Detection Transformer

Sumukh Badam
Srinidhi Kalgundi Srinivas

Motivation

- Grasp detection for robots is a highly special skill as it requires the robot to dexterously move and grasp the object
- Different from conventional object detection paradigms with emphasis on the pose of the objects in the image

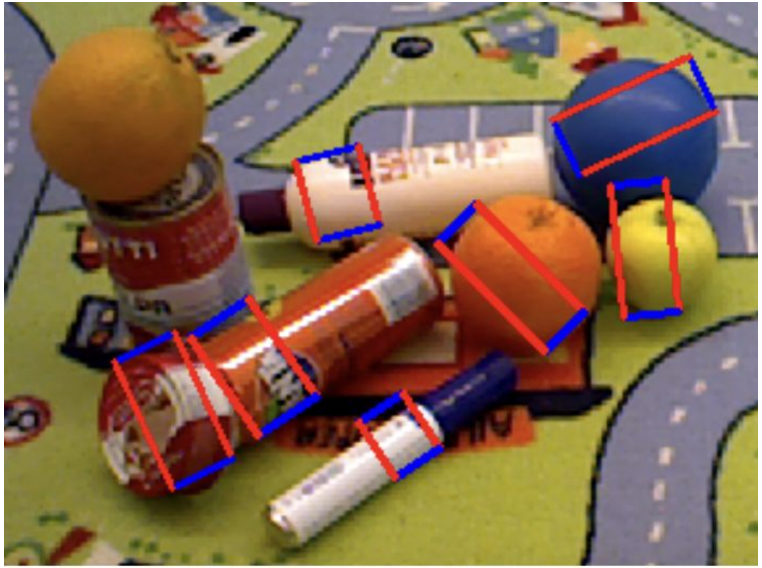


Image reference: End-to-end Trainable Deep Neural Network for Robotic Grasp Detection and Semantic Segmentation from RGB

Overview

Approach:

- Orientation parameters was considered as class labels for the images
- Transformers was used to perform object detection in the scene
- Pre-trained Resnet model was used as Encoder backbone of the network

Scope:

- Multiple objects in the scene is not considered in this project
- Use self-attention only in the decoder part of the network

Challenges:

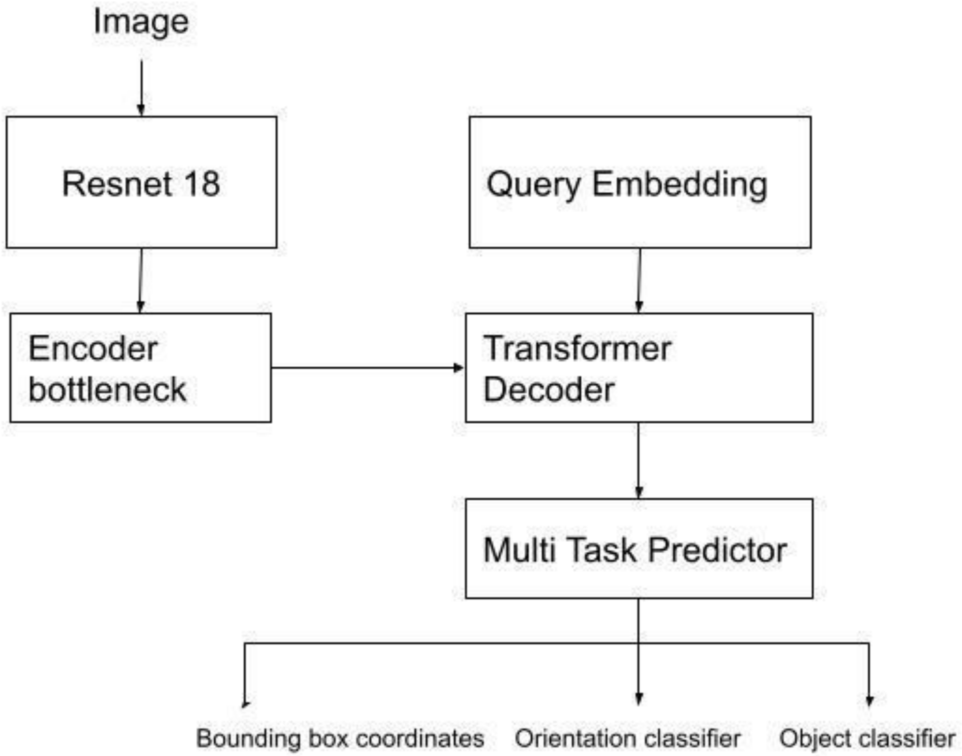
- Lack of real world datasets
- Adding orientation parameters to the bounding box is difficult
- Fine tuning and training the network as the Detection Transformers are hard to be fine tuned
- Depth information is lacking due to use of monocular camera

Detailed Architecture

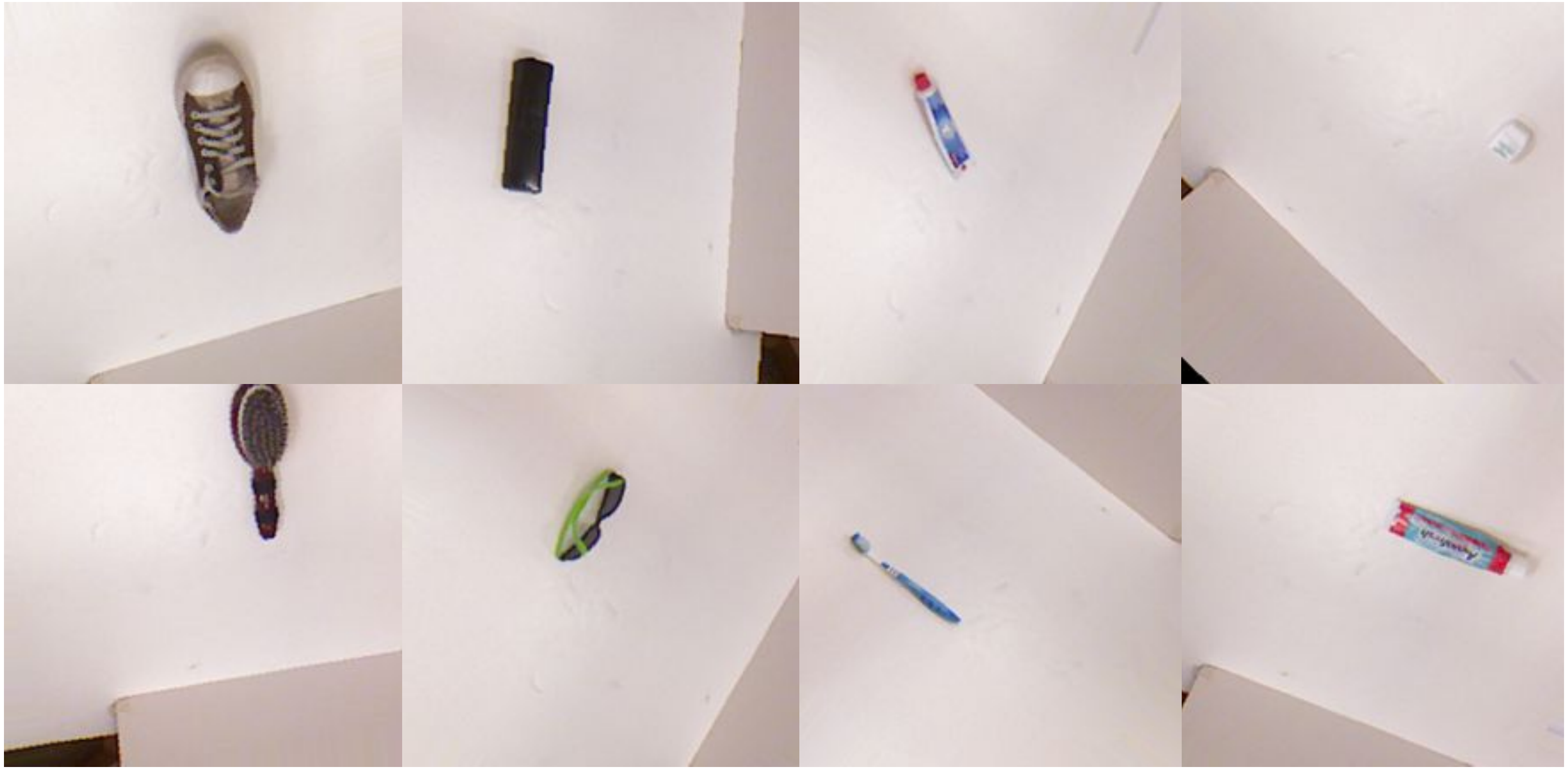
- The network follows Encoder-Decoder architecture similar to DETR[1]
- Encoder is a pre-trained ResNet-18 model without the self attention in the encoder
- Encoder bottleneck layer is used to reduced the dimensionality of the input to the decoder
- Decoder layer is a transformer layer with multi-headed attention module
- Output of the network are *n*-queries each of which provides bounding box coordinates and orientations

[1] Carison *et al.* End-to-end object detection with transformers

Architecture



Dataset



Training

- The network was trained for 500 epochs
- Initial learning rate was set to 1e-4
- Learning rate scheduler was used to reduce the learning on plateau
- Data augmentation using Rotation, Translation and Horizontal Flip was used
- ~5k images for training and 128 images for validation

Cost Function

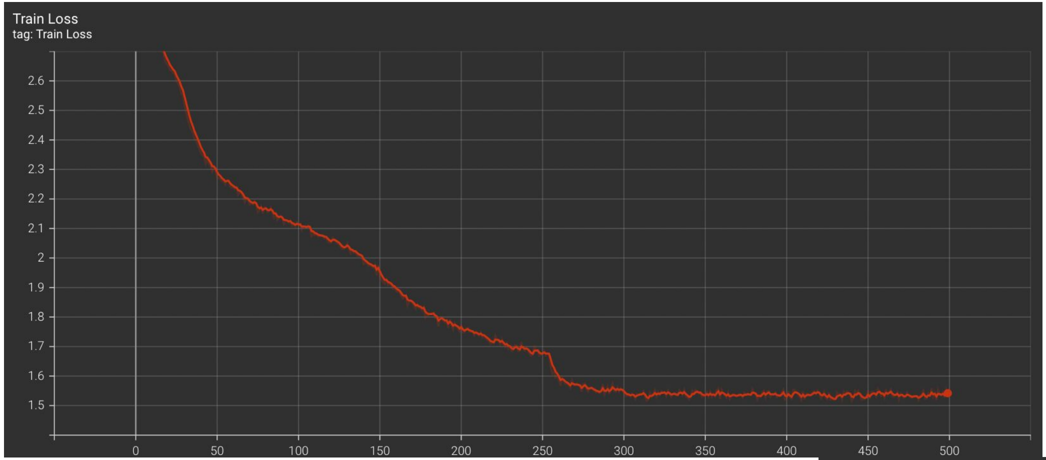
- Used to choose the 1 out of n possible prediction
    - We used 4, 8 and 16 query embedding
- Cost function is calculated using
    - L1 distance between the predicted bounding box and target bounding box
    - Generalised Intersection over Union
- Choose the argmin of the cost function
    - Each image contains only one object hence we need argmin instead of linear sum assignment used in DETR

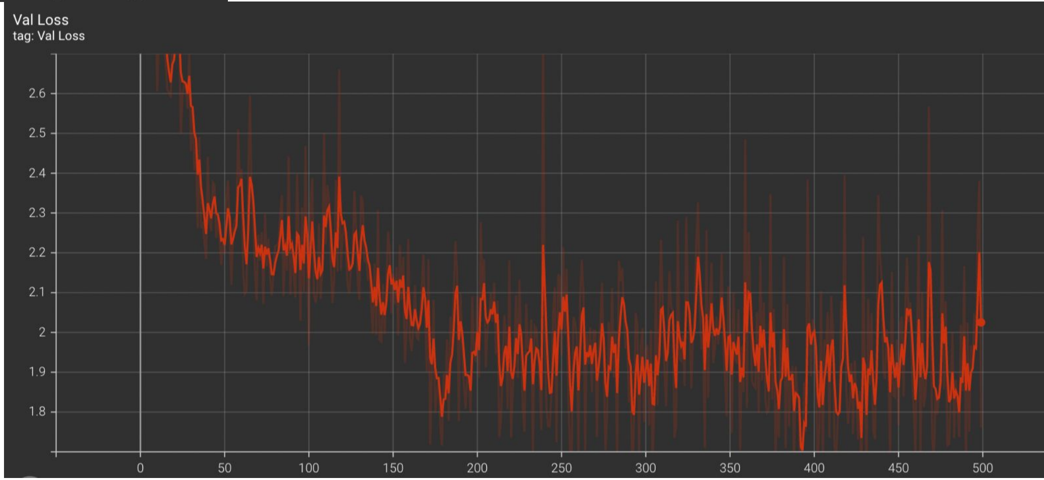
Loss Function

Bipartite Loss function similar to DETR.

- L1 distance between predicted and target bounding box
- Binary cross entropy loss used to check if the object is present in predicted bounding box
- Cross Entropy loss for classifying the orientation of bounding box
  - Resolution of 10 Degree
- Generalised Intersection over union loss
- Loss is calculated only for the selected bounding box
  - Selection of bounding box is done by taking argmin of cost function

Training


Train Loss
tag: Train Loss
2.6
2.5
2.4
2.3
2.2
2.1
2
1.9
1.8
1.7
1.6
1.5
0
50
100
150
200
250
300
350
400
450
500

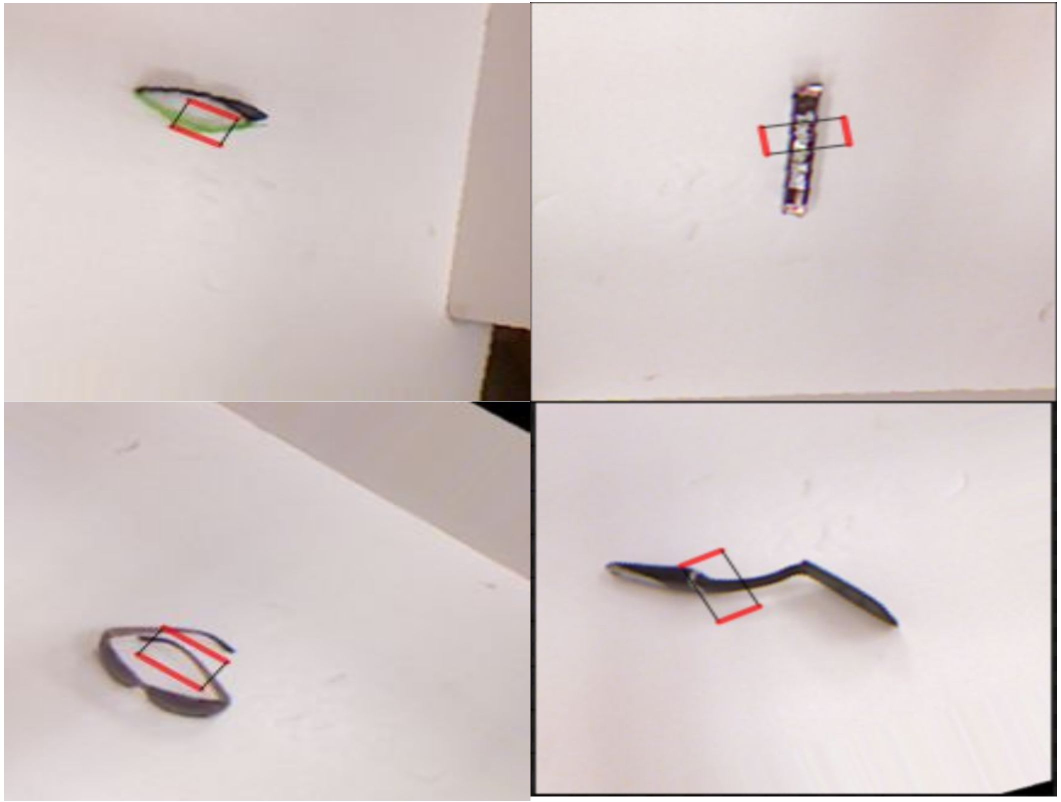

Val Loss
tag: Val Loss
2.6
2.5
2.4
2.3
2.2
2.1
2
1.9
1.8
0
50
100
150
200
250
300
350
400
450
500

Results

- For small dataset like cornell network with 4 query embedding is able to learn to predict bounding box
- Generalised IOU helps the network to predict bounding box better

Model	**Batch Size**	**Query Embeddings**	**mIoU**
Vanilla loss	32	2	0.49
Vanilla loss	64	4	0.5
Bipartite loss	32	4	0.51
Bipartite loss	32	8	0.513
Bipartite loss	32	16	0.512

Results



Multiple possible grasp

- Many possible grasps

Predicted

Target

Problem due to 2D image

- Lack of depth information in 2D image
    - Network is not able to detect edges due to lack of depth information

Target

Prediction

THANK YOU!