RAMAIAH UNIVERSITY OF APPLIED SCIENCES

Assignment

Course Code	CSE308A
Course Name	Computer Vision
Programme	B.Tech
Department	CSE
Faculty	FET

Name of the Student	Satyajit Ghana
Reg. No.	17ETCS002159
Semester/Year	07/2021
Course Leader(s)	Dr. Aruna Kumar S V

Declaration Sheet

Student Name	Satyajit Ghana		
Reg. No	17ETCS002159		
Programme	B.Tech	Semester/Year	07/2021
Course Code	CSE308A		
Course Title	Computer Vision		
Course Date		to	
Course Leader	Dr. Aruna Kumar S V		

Declaration

The assignment submitted herewith is a result of my own investigations and that I have conformed to the guidelines against plagiarism as laid out in the Student Handbook. All sections of the text and results, which have been obtained from other sources, are fully referenced. I understand that cheating and plagiarism constitute a breach of University regulations and will be dealt with accordingly.

Signature of the Student		Date	
Submission date stamp (by Examination & Assessment Section)			

Signature of the Course Leader and date	Signature of the Reviewer and date

Contents

List of Figures

1 Question 1

Solution to Question No. 1

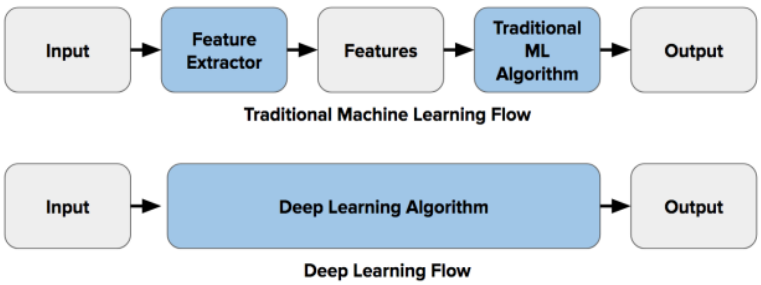
1.1 Introduction to Segmentation and Creation of Dataset

Image segmentation plays a central role in a broad range of applications, including medical image analysis autonomous vehicles, video surveillance, and augmented reality to count a few.

Numerous algorithms have been developed in literature such as thresholding, histogram-based bundling, region-growing, k-means clustering, watersheds, to more advanced algorithms like active contours, graph cuts, conditional and Markov random fields, and sparsity-based methods.

But over the past few years, segmentation has come a long way in computer vision, with advancements in deep learning, and availability of large public datasets, has resulted a paradigm shift in the field. We'll see later how well these models perform on some now standard dataset.

Quiet simply we can compare the traditional segmentation techniques and the deep learning method as below diagram,



All of the logic now resides in the Deep Learning Algorithm itself, this greatly simplifies the whole pipeline, and eliminates the need of difficult and manual labour intensive preprocessing and postprocessing steps. (Minaee et al, 2020)

Image segmentation can be formulated as a classification problem of pixels with semantic labels (semantic segmentation) or partitioning of individual objects (instance segmentation). Semantic segmentation performs pixel-level labeling with a set of object categories (e.g., human, car, tree, sky) for all image pixels, thus it is generally a harder undertaking than image classification, which predicts a single label for the entire image. Instance segmentation extends semantic segmentation scope further by detecting and delineating each object of interest in the image (e.g., partitioning of individual persons).

Below is an example of image segmentation output of a popular deep learning model, DeepLabV3

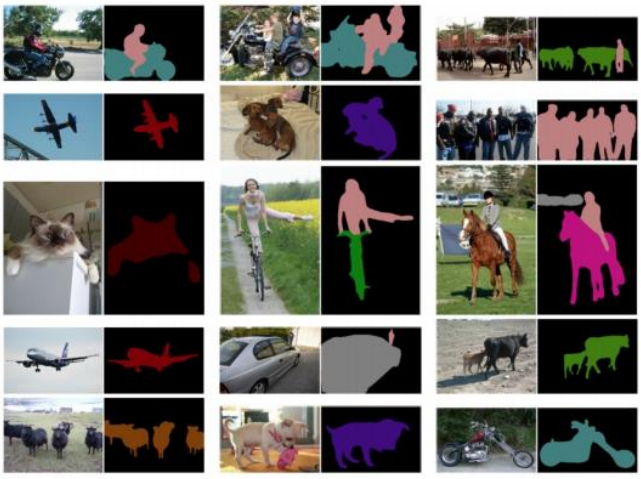


Figure 1 Segmentation results of DeepLabV3 on sample images

1.1.1 Segmentation Datasets

Datasets for segmentation are created using an annotation tool like VoTT, LabelBox or LabelMe, where each instance of the object is annotated in a specific format, we'll discuss below of the different types of available SOTA datasets used to train segmentation models.

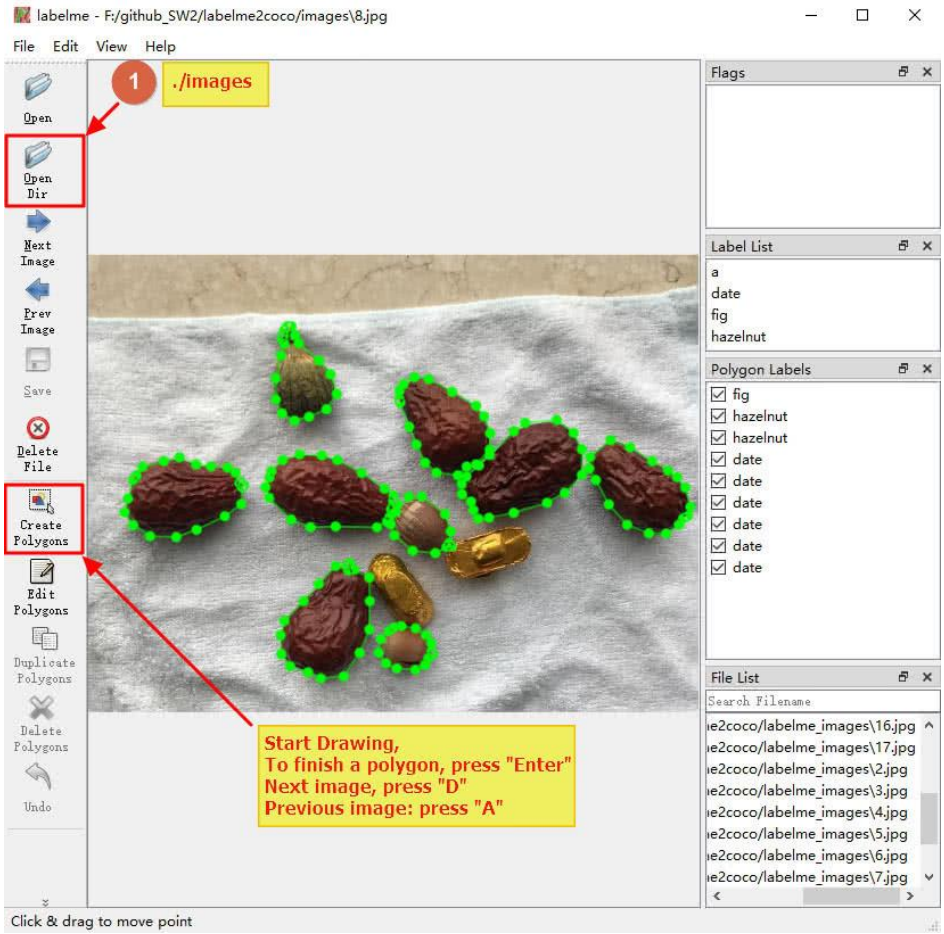


Figure 2 Example of LabelMe Dataset Annotation Tool for Segmentation

PASCAL VOC (Visual Object Classes)

This is one of the most popular datasets in CV, with annotated images available for classification, segmentation, detection, action recognition, and person layout. The dataset is divided into training and validation, with 1,464 and 1,449 images respectively.

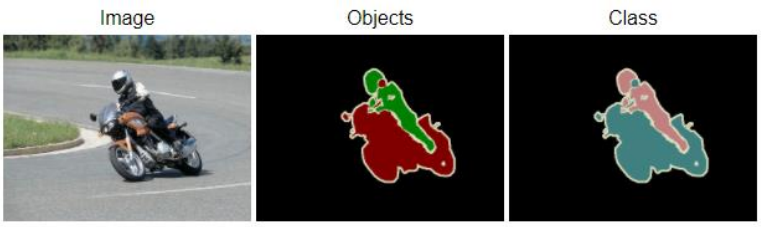
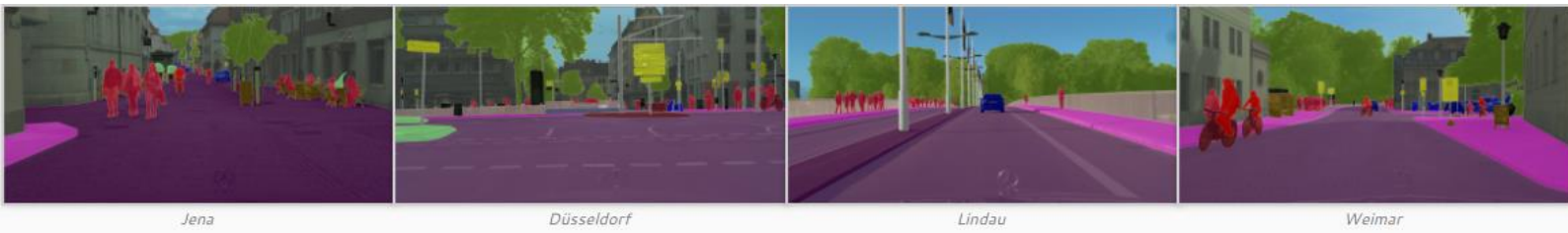


Figure 3 An example image from the PASCAL VOC dataset

Cityscapes



This is a large-scale database with a focus on semantic segmentation of urban street scenes. It contains recorded video sequences from 50 cities, with high quality pixel-level annotation of 5k frames.

1.2 Old School Image segmentation methods (deprecated)

Thresholding—divides an image into a foreground and background. A specified threshold value separates pixel into one of two levels to isolate objects. Thresholding converts grayscale images into binary images or distinguishes the lighter and darker pixels of a color image.

K-means clustering—an algorithm identifies groups in the data, with the variable K representing the number of groups. The algorithm assigns each data point (or pixel) to one of the groups based on feature similarity. Rather than analyzing predefined groups, clustering works iteratively to organically form groups.

Histogram-based image segmentation—uses a histogram to group pixels based on "gray levels". Simple images consist of an object and a background. The background is usually one gray level and is the larger entity. Thus, a large peak represents the background gray level in the histogram. A smaller peak represents the object, which is another gray level.

Edge detection—identifies sharp changes or discontinuities in brightness. Edge detection usually involves arranging points of discontinuity into curved line segments, or edges. For example, the border between a block of red and a block of blue.

1.3 Identify and explain the appropriate pre-processing techniques

Generally speaking, there's very little preprocessing involved in deep learning, most of it only includes applying image transforms so that the model does not overfit.

```
if image_set == 'train':
    return T.Compose([
        T.RandomHorizontalFlip(),
        T.RandomSelect(
            T.RandomResize(scales, max_size=1333),
            T.Compose([
                T.RandomResize([400, 500, 600]),
                T.RandomSizeCrop(384, 600),
                T.RandomResize(scales, max_size=1333),
            ])
        ),
        normalize,
    ])

if image_set == 'val':
    return T.Compose([
        T.RandomResize([800], max_size=1333),
        normalize,
    ])
```

This is an example of preprocessing done on COCO dataset before feeding it to the network on DETR Model (Carion et al, 2020), basically for training we do: Random Horizontal Flip, (Random Resize or Random Resize, Random Size Crop) and Image Normalize. `(mean=0, std=1)`.

This step is only so that we can generate more data samples from the existing dataset with varying sizes and scales, this will reduce overfitting by a lot.

1.4 Identify and explain the appropriate segmentation techniques

Below we'll discuss various SOTA Segmentation models, but specifically we'll look into two diverse kind of networks, which are 3 years apart from their papers being published, these two networks are leading SOTA segmentation models at their times.

DETR (DEtection TRansformer)

The main ingredients of the new framework, called DEtection TRansformer or DETR, are a set-based global loss that forces unique predictions via bipartite matching, and a transformer encoder-decoder architecture. Given a fixed small set of learned object queries, DETR reasons about the relations of the objects and the global image context to directly output the final set of predictions in parallel. The new model is conceptually simple and does not require a specialized library, unlike many other modern detectors. DETR demonstrates accuracy and

run-time performance on par with the well-established and highly-optimized Faster RCNN baseline on the challenging COCO object detection dataset. (Carion et al., 2020)

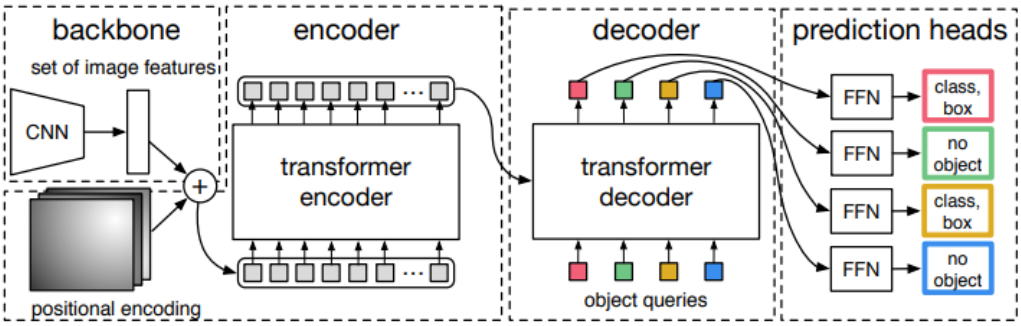


Figure 4 DETR CNN backbone

DETR uses a conventional CNN backbone to learn the 2D representation of the image, the model flattens it and supplements it with a positional encoding before passing it into a transformer encoder. A transformer decode then takes as input a small fixed number of learned positional embeddings, which are object queries, and additionally attends to the encoder output. This output embeddings are passed onto a FFN that predicts the mask like output, as described in figure below, these masks are the panoptic segmentation maps.

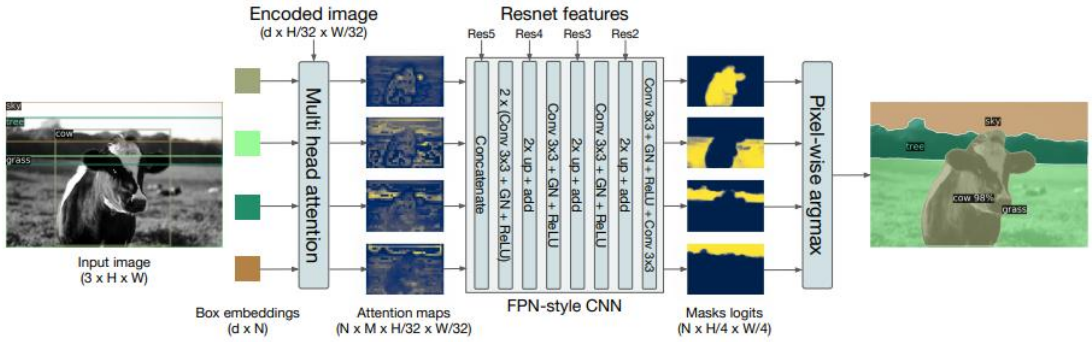


Figure 5 Illustration of the panoptic head, for panoptic image segmentation

DeepLabV3

DeepLabV1 and DeepLabV3 are among some of the most popular segmentation approaches developed by Chen et al. (2017). The main feature of their proposed network is introduction of Atrous Convolution or aka Dilated Convolution, which addressed the decreasing resolution

in the network, and also Atrous Spatial Pyramid Pooling (ASPP), which probes an incoming convolutional feature layer with filers at multiple sampling rates, thus capturing objects as well as image context at multiple scales to robustly segment objects at multiple scales.

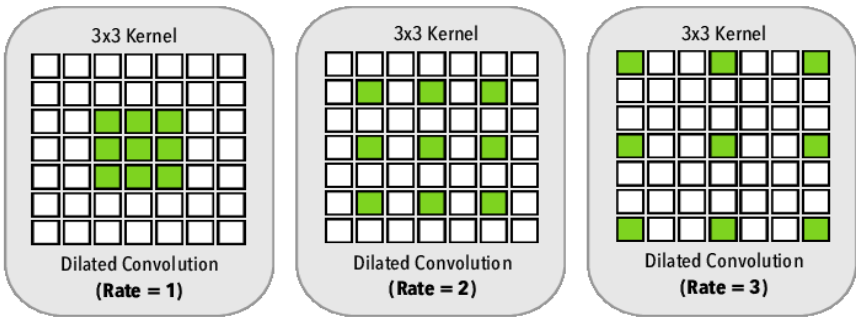


Figure 6 Dilated Convolution

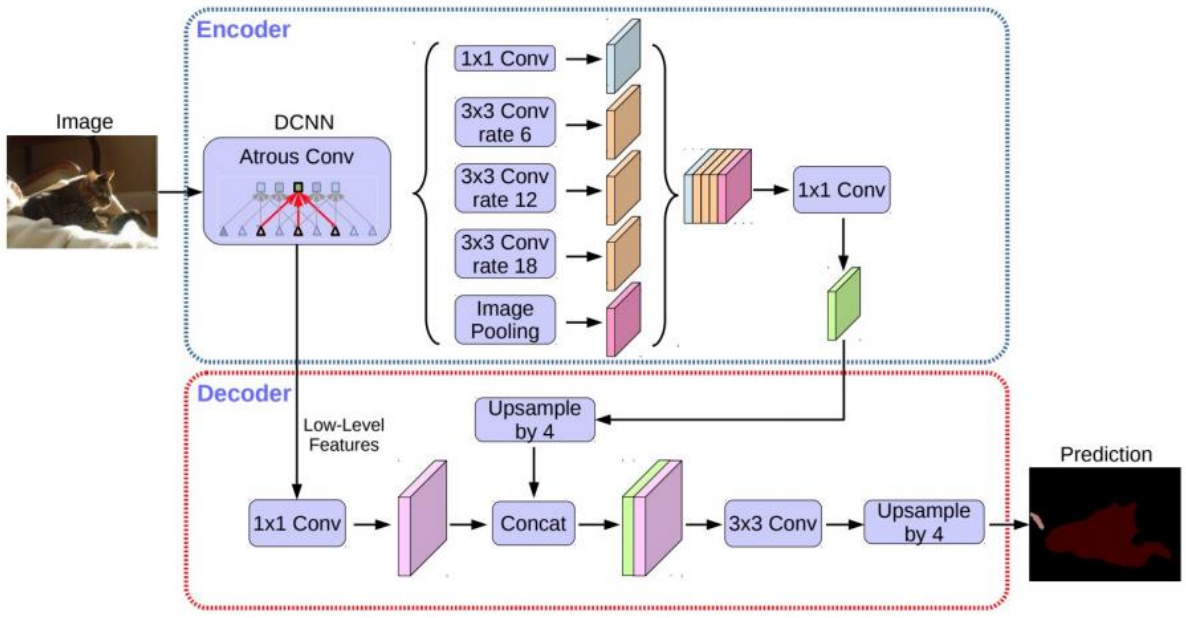


Figure 7 DeepLabV3 Model

1.5 Example Usages of Segmentation

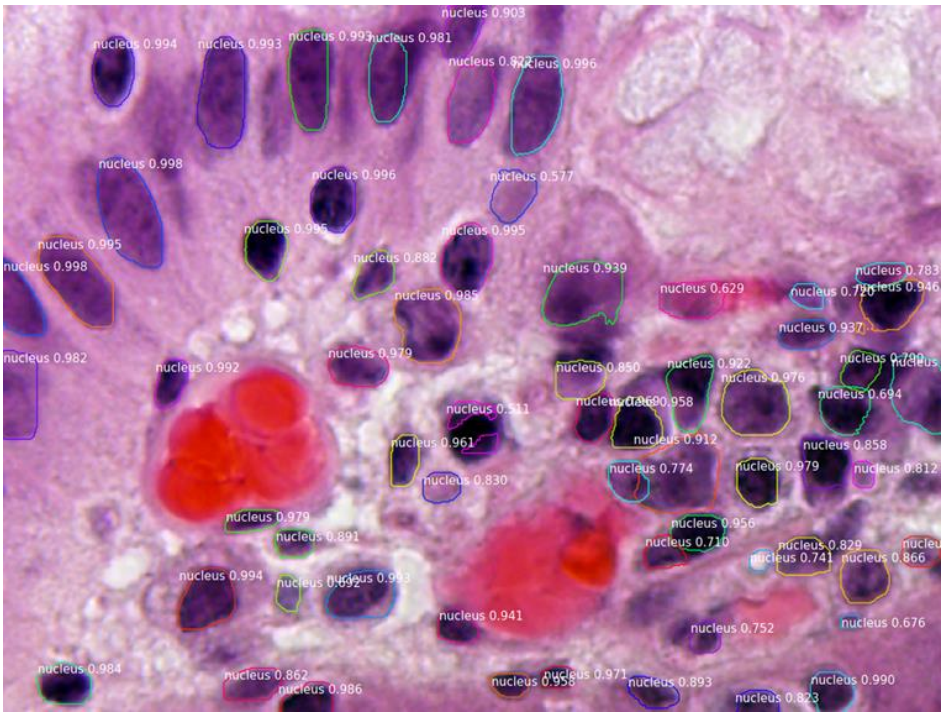


Figure 8 Segmenting Nuclei in Microscopy Images

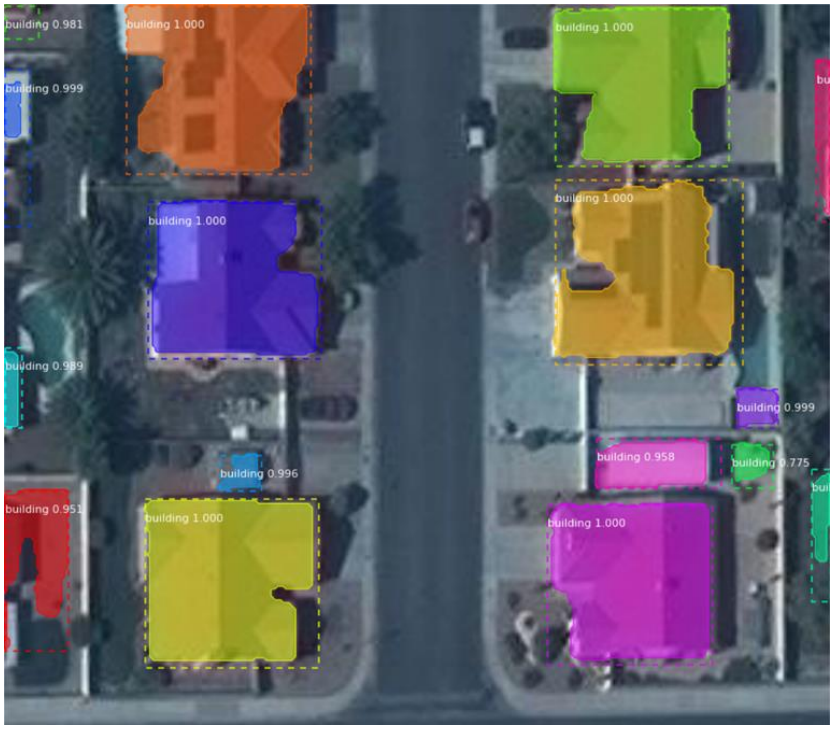


Figure 9 Convert satellite imagery to maps for use by humanitarian organizations

1.6 Sample Outputs of SOTA models

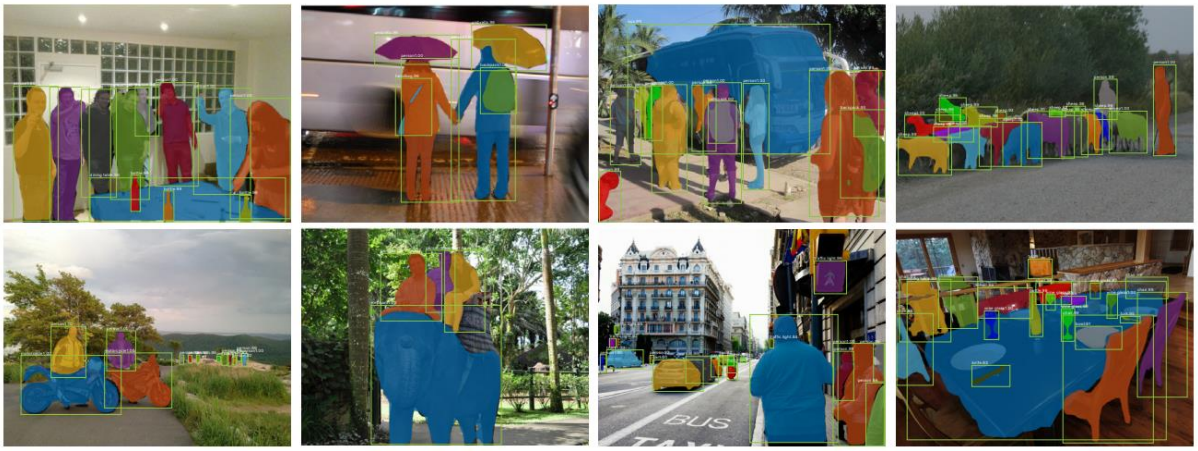


Figure 10 **Mask R-CNN** results on COCO test dataset

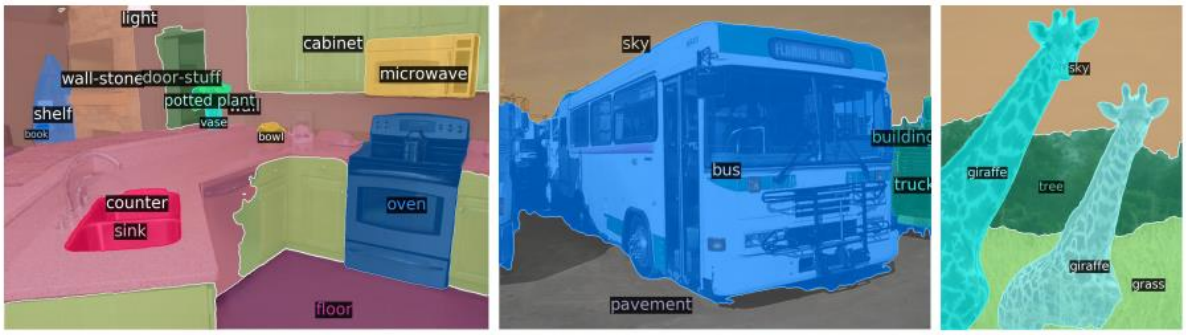


Figure 12 Panoptic Segmentation generated by **DETR-R101** on COCO dataset

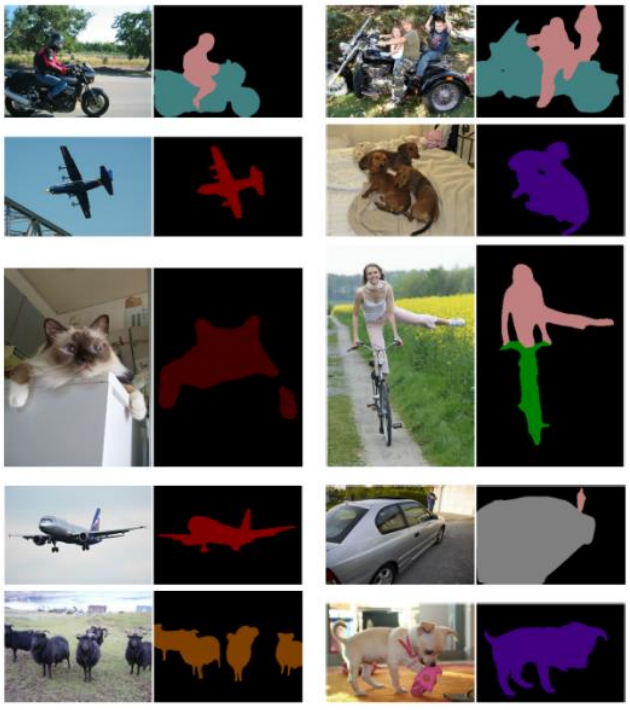


Figure 11 **DeepLabV3** on PASCAL VOC Dataset

2 Question 2

Solution to Question 2

2.1 Perform pre-processing on the images of the created dataset

For the sake of keeping the assignment simple and doable in a short period of time, a small dataset of Cats and Dogs was to be created, the images where collected from flickr (Public Image Website). In total 32 Images of Images containing Cats and Dogs was collected.

VoTT was used to annotate the dataset with their respective segment polygons, this was then exported to VOC format, which was then converted to COCO JSON format.

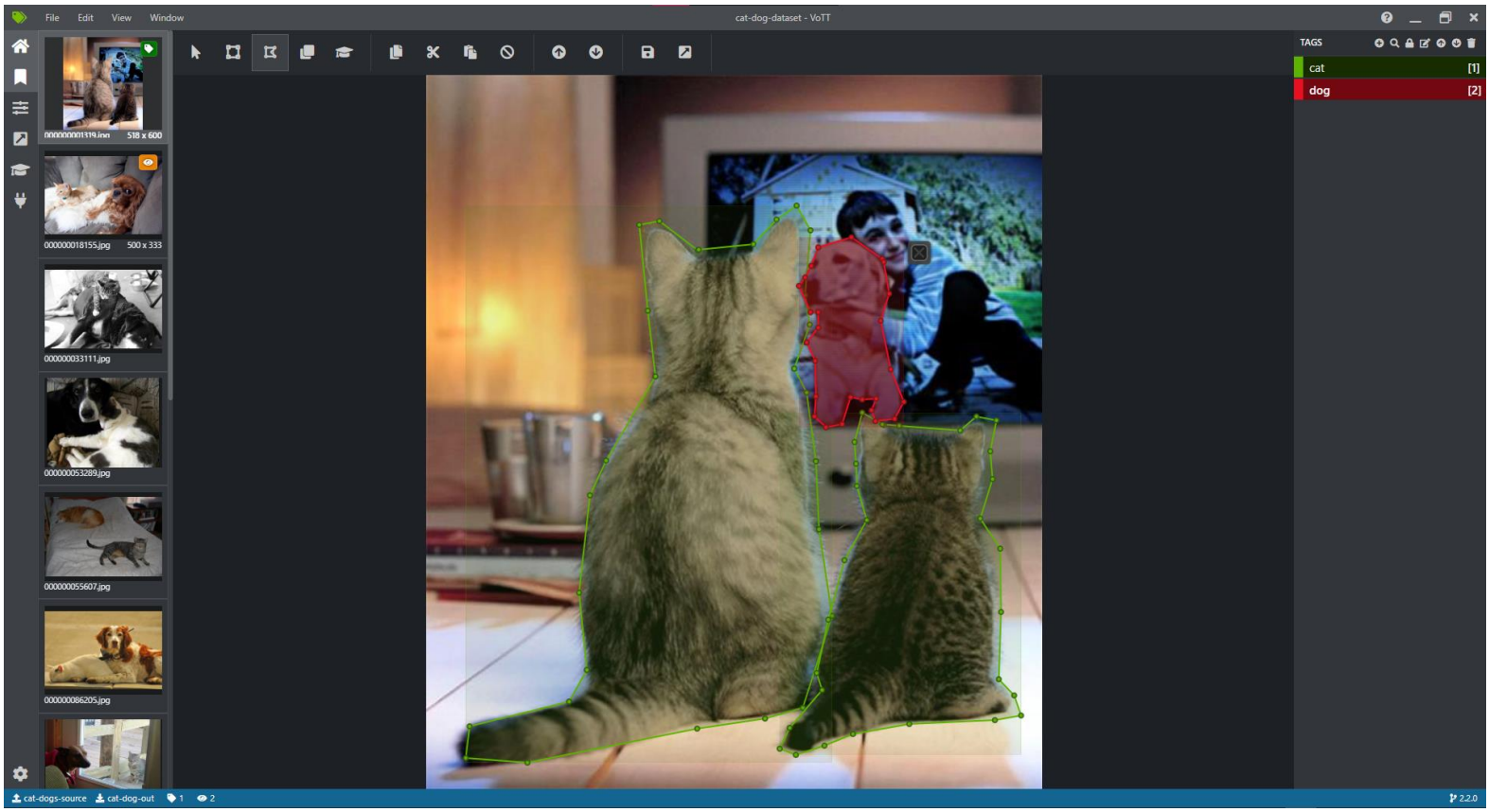


Figure 13 Annotating Cat-Dog Dataset

Below is a very brief example of the COCO style annotations for the Cat-Dog Dataset, which contain the polygon coordinates for each of the segment annotation and also the bbox coordinates.

```
{
  "images": [
    {
      "file_name": "000000183757.jpg",
      "height": 640,
      "width": 480,
      "id": 183757
    },
    . . .
  ],
  "annotations": [
    {
      "segmentation": [
        [ 229.48, 297.46, 200.05, 314.7, 198.99, 325.21, 212.58, 361.54, 218.13, . . .]
      ],
      "area": 60228.7593,
      "iscrowd": 0,
      "image_id": 283471,
      "bbox": [
        119.95,
        151.12,
        425.1,
        274.49
      ],
      "category_id": 2,
      "id": 2471
    },
    . . .
  ],
  "categories": [
    {
      "supercategory": "animal",
      "id": 1,
      "name": "cat"
    },
    {
      "supercategory": "animal",
      "id": 2,
      "name": "dog"
    }
  ]
}
```


Futher, the following are the pre-processing steps done on the images before feeding it to the model.

- `ResizeShortestEdge(`
  `short_edge_length=(640, 672, 704, 736, 768, 800), max_size=1333`
  `): Scale the shorter edge to the given size, with a limit of `max_size` on the longer edge. If `max_size` is reached, then downscale so that the longer edge does not exceed max_size.`

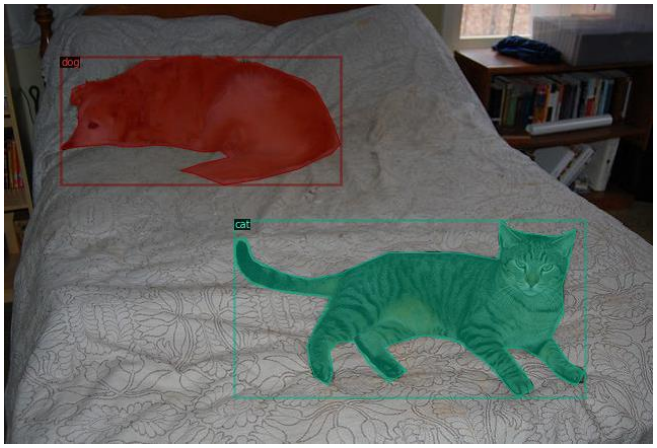
- `RandomFlip(prob=0.5): Flip the image horizontally or vertically with the given probability.`

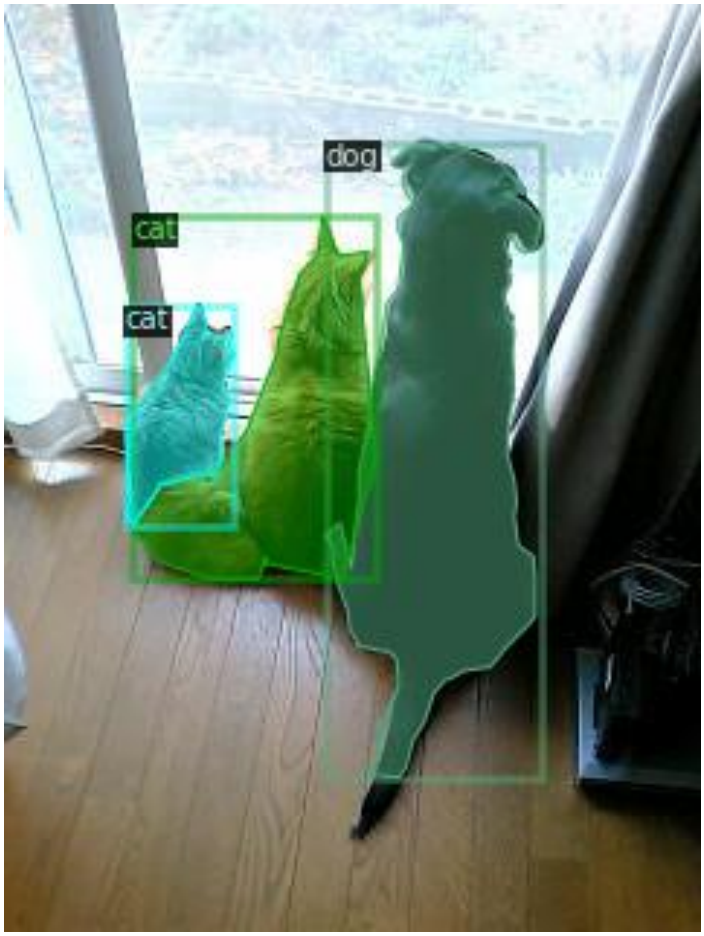
2.1.1 Image Samples Visualized using COCO Tools

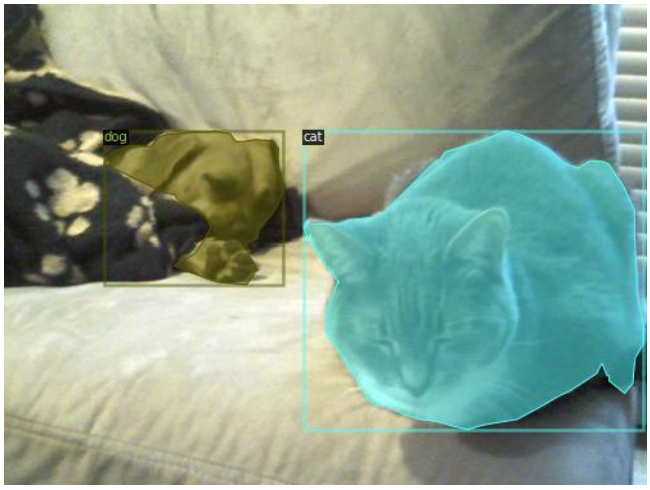
To visualize the annotation and also to make sure that the annotations are correct and in the right COCO format, `pycocotools` was used.

```
COCO categories:
cat dog

COCO supercategories:
animal
```


dog
cat


dog
cat
cat


dog
cat


cat
dog

2.2 Training the model

Before we can perform segmentation, we need to train the model, for this detectron2 framework was used to train Mask RCNN with ResNet 50 as the backbone, the details of the model is discussed in 2.3.

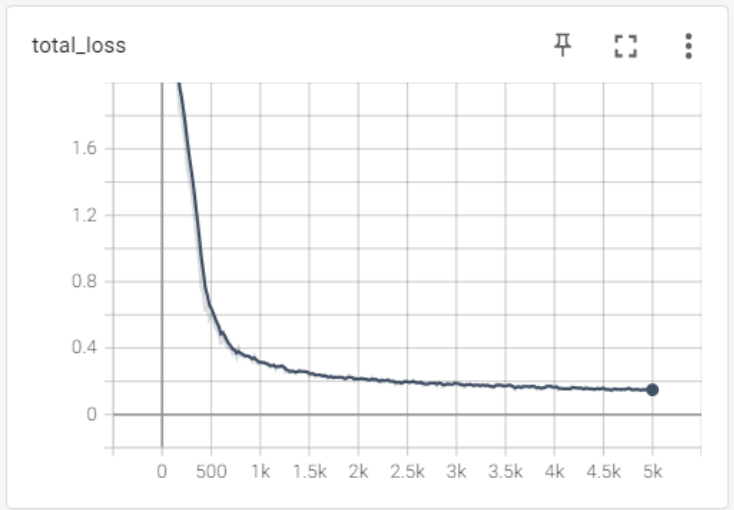
The Model (in PyTorch)

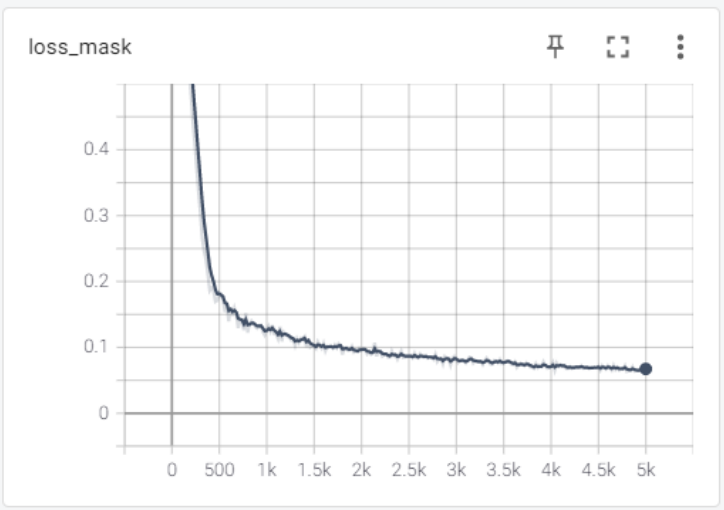
```
GeneralizedRCNN(
  (backbone): FPN(
    (fpn_lateral2): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (fpn_lateral3): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (fpn_lateral4): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output4): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (fpn_lateral5): Conv2d(2048, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output5): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (top_block): LastLevelMaxPool()
    (bottom_up): ResNet(
      (stem): BasicStem(
        (conv1): Conv2d(
          3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False
          (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
        )
      )
---------  Resnet50 BACKBONE  ------
  (proposal_generator): RPN(
    (rpn_head): StandardRPNHead(
      (conv): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (objectness_logits): Conv2d(256, 3, kernel_size=(1, 1), stride=(1, 1))
      (anchor_deltas): Conv2d(256, 12, kernel_size=(1, 1), stride=(1, 1))
    )
    (anchor_generator): DefaultAnchorGenerator(
      (cell_anchors): BufferList()
    )
  )
  (roi_heads): StandardROIHeads(
    (box_pooler): ROIPooler(
      (level_poolers): ModuleList(
        (0): ROIAlign(output_size=(7, 7), spatial_scale=0.25, sampling_ratio=0, aligned=True)
        (1): ROIAlign(output_size=(7, 7), spatial_scale=0.125, sampling_ratio=0, aligned=True)
        (2): ROIAlign(output_size=(7, 7), spatial_scale=0.0625, sampling_ratio=0, aligned=True)
        (3): ROIAlign(output_size=(7, 7), spatial_scale=0.03125, sampling_ratio=0, aligned=True)
      )
    )
    (box_head): FastRCNNConvFCHead(
      (flatten): Flatten(start_dim=1, end_dim=-1)
      (fc1): Linear(in_features=12544, out_features=1024, bias=True)
      (fc_relu1): ReLU()
      (fc2): Linear(in_features=1024, out_features=1024, bias=True)
      (fc_relu2): ReLU()
    )
    (box_predictor): FastRCNNOutputLayers(
      (cls_score): Linear(in_features=1024, out_features=3, bias=True)
      (bbox_pred): Linear(in_features=1024, out_features=8, bias=True)
    )
```

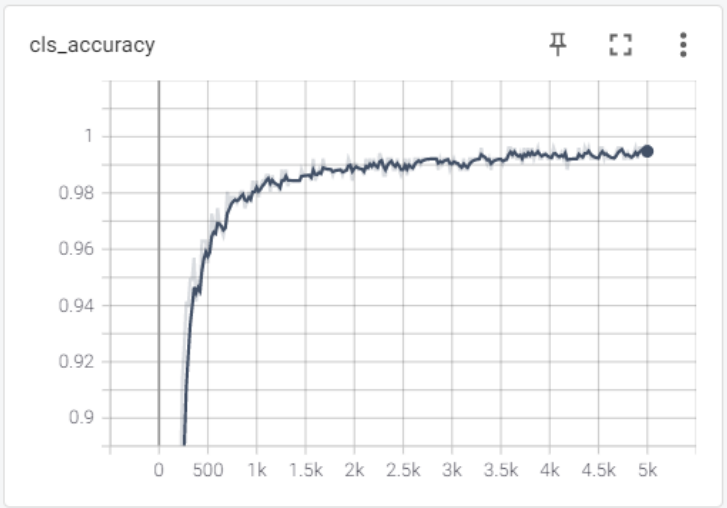
```
    (mask_pooler): ROIPooler(
      (level_poolers): ModuleList(
        (0): ROIAlign(output_size=(14, 14), spatial_scale=0.25, sampling_ratio=0, aligned=True)
        (1): ROIAlign(output_size=(14, 14), spatial_scale=0.125, sampling_ratio=0, aligned=True)
        (2): ROIAlign(output_size=(14, 14), spatial_scale=0.0625, sampling_ratio=0, aligned=True)
        (3): ROIAlign(output_size=(14, 14), spatial_scale=0.03125, sampling_ratio=0, aligned=True)
      )
    )
    (mask_head): MaskRCNNConvUpsampleHead(
      (mask_fcn1): Conv2d(
        256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)
        (activation): ReLU()
      )
      (mask_fcn2): Conv2d(
        256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)
        (activation): ReLU()
      )
      (mask_fcn3): Conv2d(
        256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)
        (activation): ReLU()
      )
      (mask_fcn4): Conv2d(
        256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)
        (activation): ReLU()
      )
      (deconv): ConvTranspose2d(256, 256, kernel_size=(2, 2), stride=(2, 2))
      (deconv_relu): ReLU()
      (predictor): Conv2d(256, 2, kernel_size=(1, 1), stride=(1, 1))
    )
  )
)
```

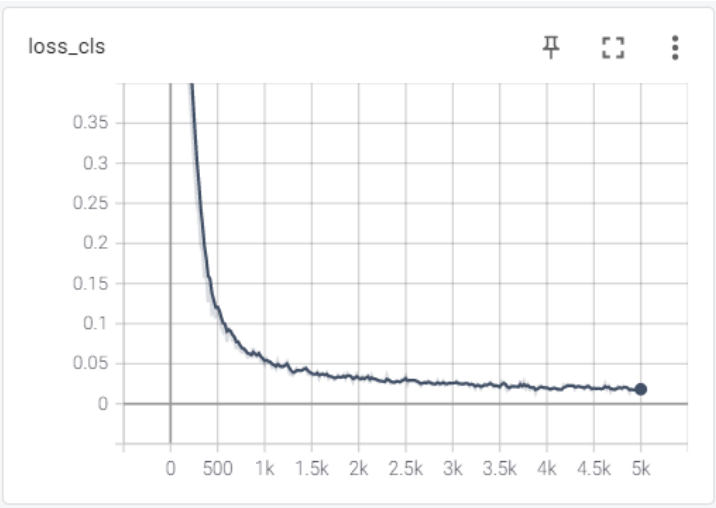
The Model was trained for 5000 Iteration on Nvidia Tesla P100 in 40 mins total

Model Metrics









Model Evaluation

The model was evaluated on a test cat-dog dataset, which gave Average Precision of 97.69%

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.977
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.985
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.985
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.970
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.921
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.979
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.979
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.973
[01/17 11:05:02 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50  |  AP75  |  APs  |   APm   |  APl   |
|:------:|:------:|:------:|:-----:|:-------:|:------:|
| 97.690 | 98.515 | 98.515 |  nan  | 100.000 | 97.012 |
[01/17 11:05:02 d2.evaluation.coco_evaluation]: Per-category bbox AP:
| category   | AP     | category   | AP     |
|:-----------|:-------|:-----------|:-------|
| cat        | 96.679 | dog        | 98.702 |
```

2.3 Perform Segmentation, Results and Discussions

The segmentation now can be done simply by pre-processing the image, and passing it to the model, the result is the segmentation masks, which then can be further overlayed over the

original image for some nice visualization. All of the visualization code was taken from detectron2 framework, with some minor modifications.

```
with torch.no_grad():  # https://github.com/sphinx-doc/sphinx/issues/4258
    # Apply pre-processing to image.
    if self.input_format == "RGB":
        # whether the model expects BGR inputs or RGB
        original_image = original_image[:, :, ::-1]
    height, width = original_image.shape[:2]
    image = self.aug.get_transform(original_image).apply_image(original_image)
    image = torch.as_tensor(image.astype("float32").transpose(2, 0, 1))

    inputs = {"image": image, "height": height, "width": width}
    predictions = self.model([inputs])[0]
    return predictions
```

Figure 14 Code for running segmentation on an input image (detectron2)

2.3.1 Discussion on the Model

Mask R-CNN is an extension to Faster-RCNN, it simply adds a branch for predicting segmentation masks on each Region of Interest (RoI), in parallel with the existing branch for classification and bounding box regression. The mask branch is small FCN (Fully Convolutional Network) applied to each RoI, predicting a segmentation mask in pixel-to-pixel manner. Faster R-CNN consists of two stages. The first stage, called a Region Proposal Network (RPN), proposes candidate object bounding boxes. The second stage, which is in essence Fast R-CNN [9], extracts features using RoIPool from each candidate box and performs classification and bounding-box regression. The features used by both stages can be shared for faster inference. (He et al, 2017)

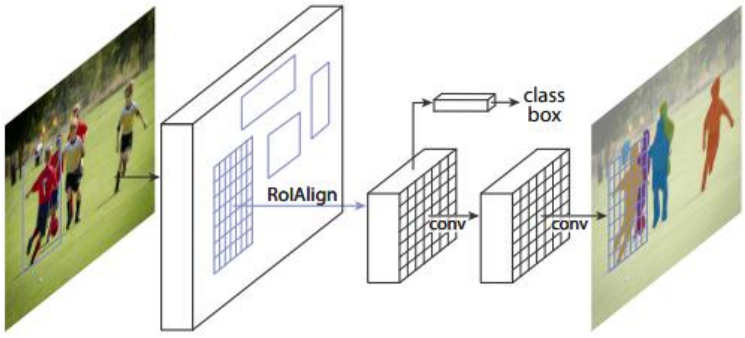


Figure 15 The Mask R-CNN framework for instance segmentation

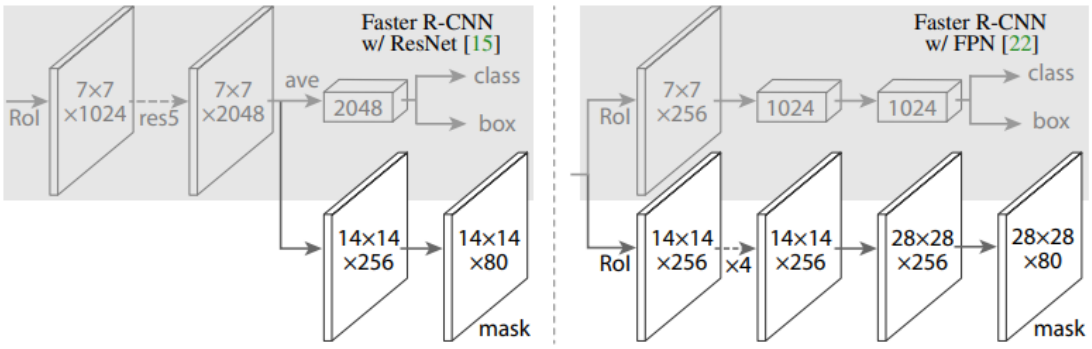
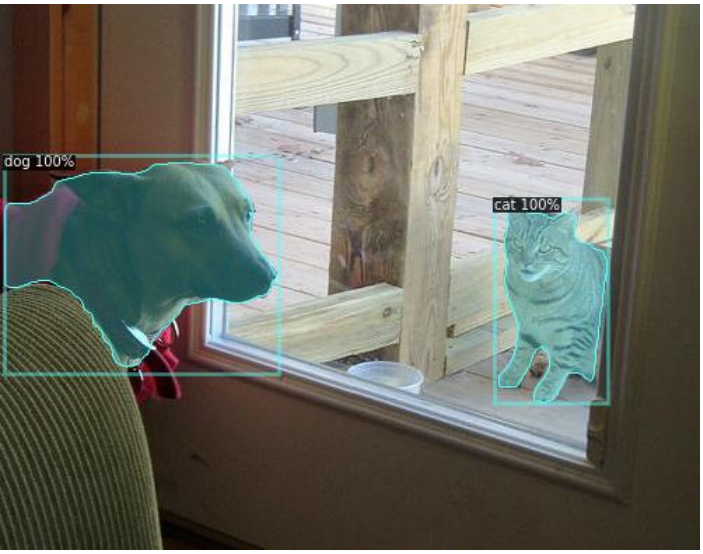


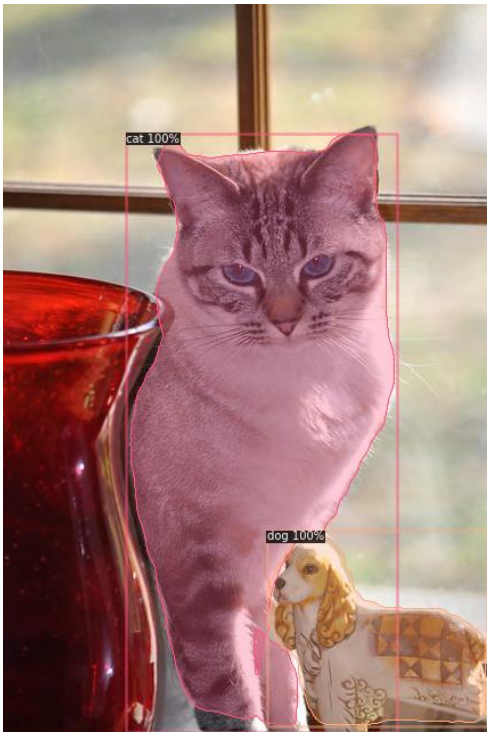
Figure 16 Head Architecture of Mask RCNN extended from Faster RCNN (He et al. 2017)

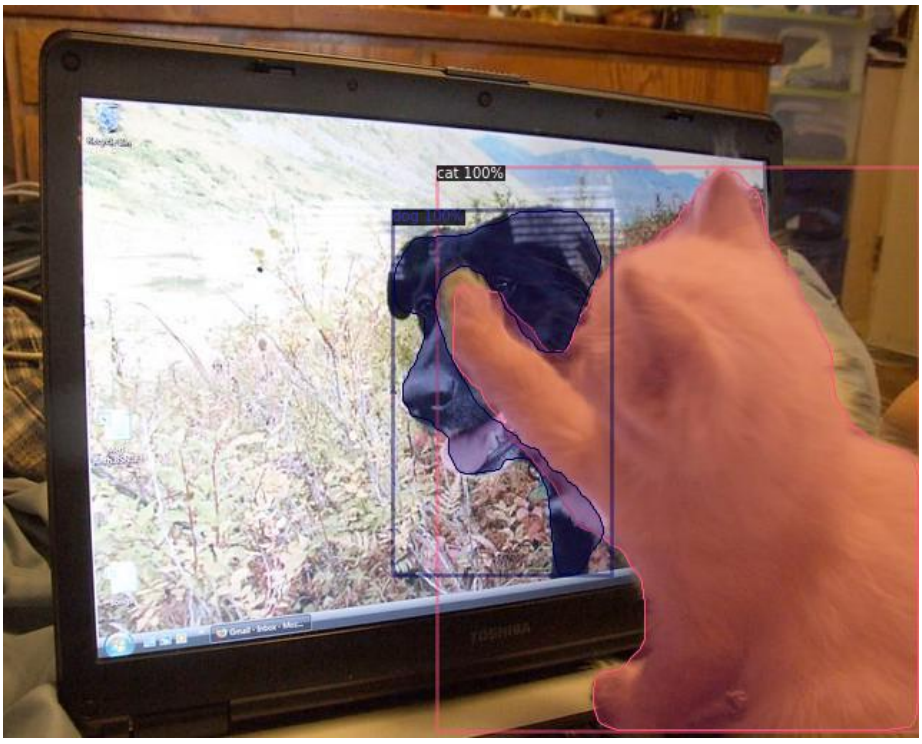
2.3.2 Test Images Prediction

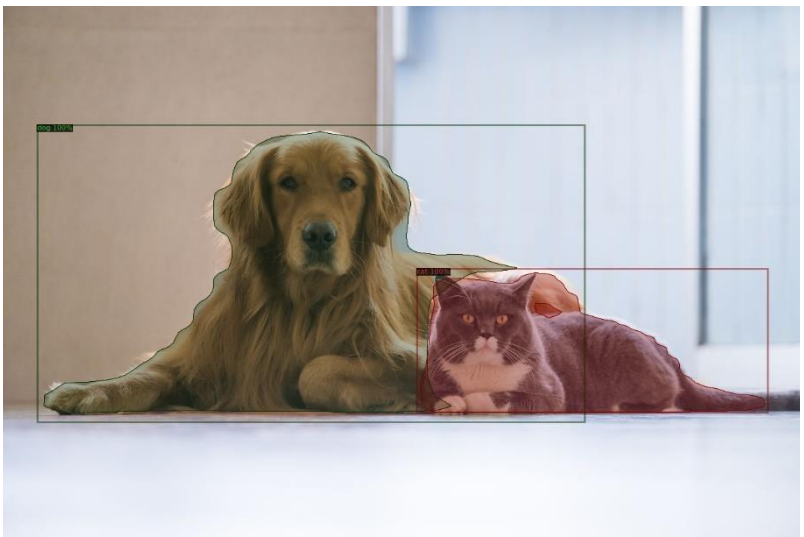
To test the model, few images were taken from google images that contain our target classes, and the model was run on these. The outputs are as follows.


dog 100%
cat 100%


cat 100%
dog 100%

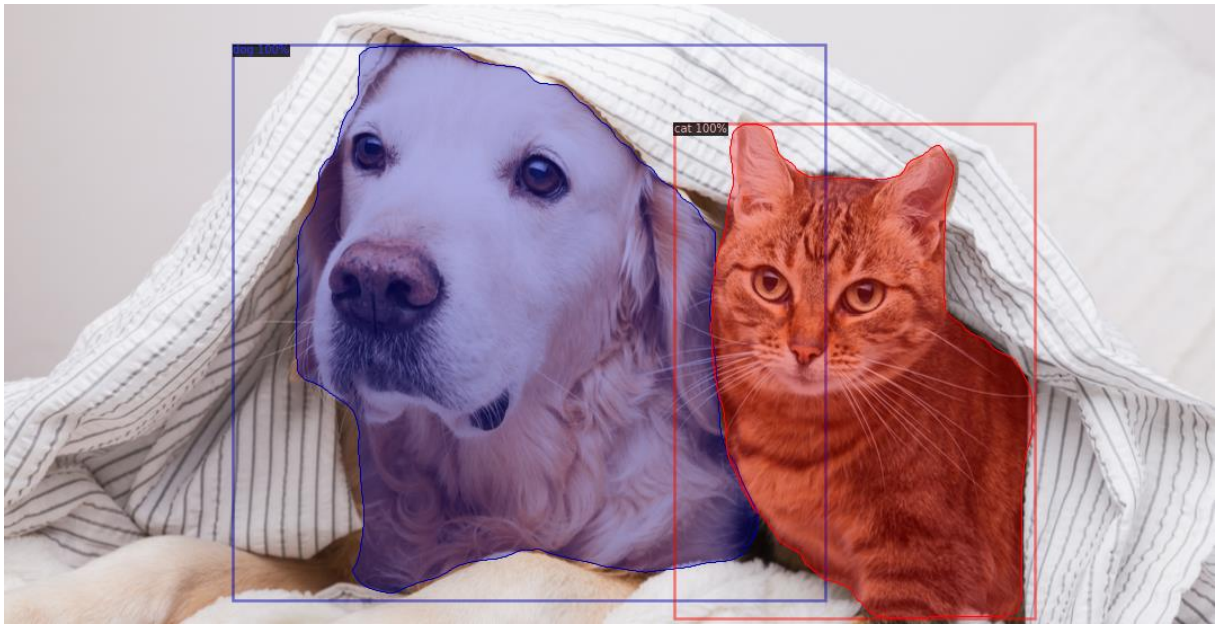

cat 100%
dog 100%


cat 100%




dog 100%
cat 100%


dog 100%
cat 100%


dog 100%
cat 100%

Bibliography

1. Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A. and Zagoruyko, S., 2020. End-to-End Object Detection with Transformers. arXiv preprint arXiv:2005.12872.

2. He, K., Gkioxari, G., Dollár, P. and Girshick, R., 2017. Mask r-cnn. In Proceedings of the IEEE international conference on computer vision (pp. 2961-2969).

3. Minaee, S., Boykov, Y., Porikli, F., Plaza, A., Kehtarnavaz, N. and Terzopoulos, D., 2020. Image segmentation using deep learning: A survey. arXiv preprint arXiv:2001.05566.

4. Chen, L.C., Papandreou, G., Schroff, F. and Adam, H., 2017. Rethinking atrous convolution for semantic image segmentation. arXiv preprint arXiv:1706.05587.

5. Chen, L.C., Zhu, Y., Papandreou, G., Schroff, F. and Adam, H., 2018. Encoder-decoder with atrous separable convolution for semantic image segmentation. In Proceedings of the European conference on computer vision (ECCV) (pp. 801-818).

6. https://github.com/cocodataset/cocoapi

7. https://github.com/facebookresearch/detectron2

Appendix

Training Logs

```
[01/17 10:11:52 d2.data.datasets.coco]: Loaded 32 images in COCO format from
/content/cat_dog_coco/annotations/annotations.json
[01/17 10:11:52 d2.data.dataset_mapper]: [DatasetMapper] Augmentations used in training:
[ResizeShortestEdge(short_edge_length=(640, 672, 704, 736, 768, 800), max_size=1333,
sample_style='choice'), RandomFlip()]
[01/17 10:11:52 d2.engine.train_loop]: Starting training from iteration 0
[01/17 10:12:02 d2.utils.events]: eta: 0:40:41 iter: 19 total_loss: 2.53 loss_cls: 0.8805
loss_box_reg: 0.9396 loss_mask: 0.6926 loss_rpn_cls: 0.006285 loss_rpn_loc: 0.006694 time:
0.4791 data_time: 0.0163 lr: 4.9953e-06 max_mem: 3255M
[01/17 10:12:12 d2.utils.events]: eta: 0:40:31 iter: 39 total_loss: 2.502 loss_cls: 0.8392
loss_box_reg: 0.9773 loss_mask: 0.6875 loss_rpn_cls: 0.008476 loss_rpn_loc: 0.007306 time:
0.4786 data_time: 0.0067 lr: 9.9902e-06 max_mem: 3255M
. . . . . .
[01/17 10:51:50 d2.utils.events]: eta: 0:00:19 iter: 4959 total_loss: 0.1519 loss_cls: 0.01956
loss_box_reg: 0.0574 loss_mask: 0.07078 loss_rpn_cls: 9.858e-06 loss_rpn_loc: 0.002456 time:
0.4828 data_time: 0.0082 lr: 0.00025 max_mem: 3394M
[01/17 10:52:00 d2.utils.events]: eta: 0:00:09 iter: 4979 total_loss: 0.1468 loss_cls: 0.01878
loss_box_reg: 0.05593 loss_mask: 0.06667 loss_rpn_cls: 1.776e-05 loss_rpn_loc: 0.002517 time:
0.4828 data_time: 0.0080 lr: 0.00025 max_mem: 3394M
[01/17 10:52:13 d2.utils.events]: eta: 0:00:00 iter: 4999 total_loss: 0.1479 loss_cls: 0.01777
loss_box_reg: 0.05872 loss_mask: 0.06731 loss_rpn_cls: 2.385e-05 loss_rpn_loc: 0.003065 time:
0.4829 data_time: 0.0067 lr: 0.00025 max_mem: 3394M
[01/17 10:52:13 d2.engine.hooks]: Overall training speed: 4998 iterations in 0:40:13 (0.4829 s /
it)
[01/17 10:52:13 d2.engine.hooks]: Total training time: 0:40:19 (0:00:06 on hooks)
```