Summer Internship Project Report

Real-Time Crack Detection and Segmentation

Author
Bhuvnesh Sahu

Mentor
Prasun Agarwal

Title – Finding Cracks and other anomalies in 3D printed material.

Description – The client is a company that makes residential buildings using 3D printing technology. The usual practice is to use 3D printing machines to print various components of the house and join them to make the complete structure. Our aim is to detect whether there are cracks or other anomalies in the house structure.

We will receive the data in the form of images or video.

We will perform real-time inference on this data and predict whether cracks or other anomalies are present or not.

Solution-

- Dataset consists of the images of surfaces (walls, roof, concrete, plaster) with cracks.
- I will be using various open source models.
- Will do a comparative analysis of all the models based on inference time and accuracy score to finalize a model.
- Will try to deploy the model according to the needs of the client or as instructed by the mentor.

Week 1

Understanding Computer Vision –

- Object Detection

- Read Github Repositary of Ultralytics to understand the working of YOLO Framework

Project to understand Object Detection -

Problem statement- Real time objects Detection

Dataset- http://host.robots.ox.ac.uk/pascal/VOC/voc2009/VOCtrainval_11-May-2009.tar

- Dataset contains images and the HTML files for each image
- The HTML file contains the information about the objects in the image (Class, dimensions of the bounding boxes)
- There are 20 classes in the dataset ('person', 'car', 'chair', 'bottle', 'pottedplant', 'bird', 'dog', 'sofa', 'bicycle', 'horse', 'boat', 'motorbike', 'cat', ‘tv monitor', 'cow', 'sheep', 'aeroplane', 'train', 'diningtable', 'bus’)

Steps-

Data Preparation -

- Extract information from the HTML document into a text file
- Make train and val folders each contains some images and a text files.
- Normalize the dimensions of the bounding boxes (YOLO)

	filename	width	height	name	xmin	xmax	ymin	ymax	center_x	center_y	w	h	id
1	2007_000032.jpg	500	281	aeroplane	104	375	78	183	0.479	0.464413	0.542	0.373665	16
2	2007_000032.jpg	500	281	aeroplane	133	197	88	123	0.330	0.375445	0.128	0.124555	16
3	2007_000032.jpg	500	281	person	195	213	180	229	0.408	0.727758	0.036	0.174377	0
4	2007_000032.jpg	500	281	person	26	44	189	238	0.070	0.759786	0.036	0.174377	0
5	2007_000033.jpg	500	366	aeroplane	9	499	107	263	0.508	0.505464	0.980	0.426230	16
6	2007_000033.jpg	500	366	aeroplane	421	482	200	226	0.903	0.581967	0.122	0.071038	16

Conversion formula

- $center\_x = \frac{\frac{x_{min}+x_{max}}{2}}{width\ of\ the\ image}$
- $center\_y = \frac{\frac{y_{min}+y_{max}}{2}}{height\ of\ the\ image}$
- $w = \frac{x_{max}-x_{min}}{width\ of\ the\ image}$
- $h = \frac{y_{max}-y_{min}}{height\ of\ the\ image}$

Training

- Make a YAML file that contains nc (number of classes), names (name of classes), train (train data path), val (test data path)
- Clone github repo of YOLO v5 (Also train yolo model by installing ultralytics library)
- Use CLI command to train and export the model
  - !python train.py --data data.yaml --cfg yolov5s.yaml --batch-size 8 --name Model --epochs 50
  - !python export.py --weights runs/train/Model/weights/best.pt --include onnx --simplify --opset 12
- We have exported the model in ONNX format which can work with OpenCV.

Predictions-

- Load YOLO model using OpenCV and make a object of it.
- Write a function that takes a image and convert it into 640*640 image.
- Use the model object to predict the modified image.
- The prediction from model will contain N rows and 25 columns (N is the number of objects in the image).
- Use this information to make bounding boxes around the objects and label them.

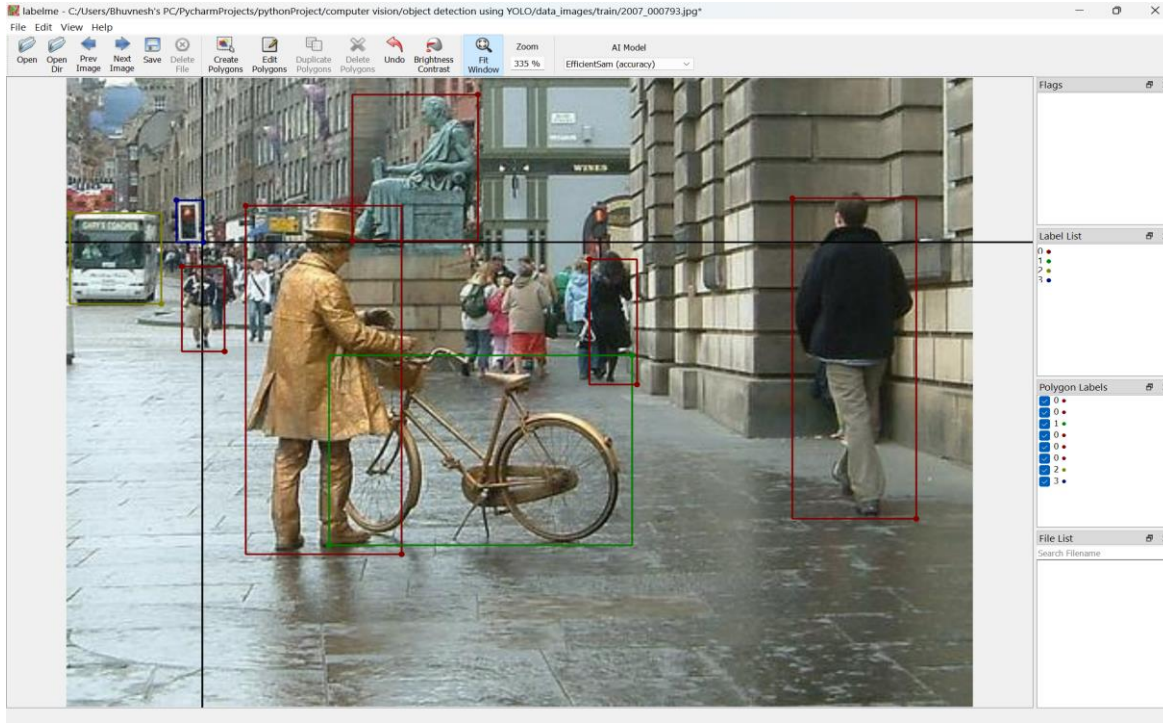
YOLO output

25 columns

5 columns | 20 columns

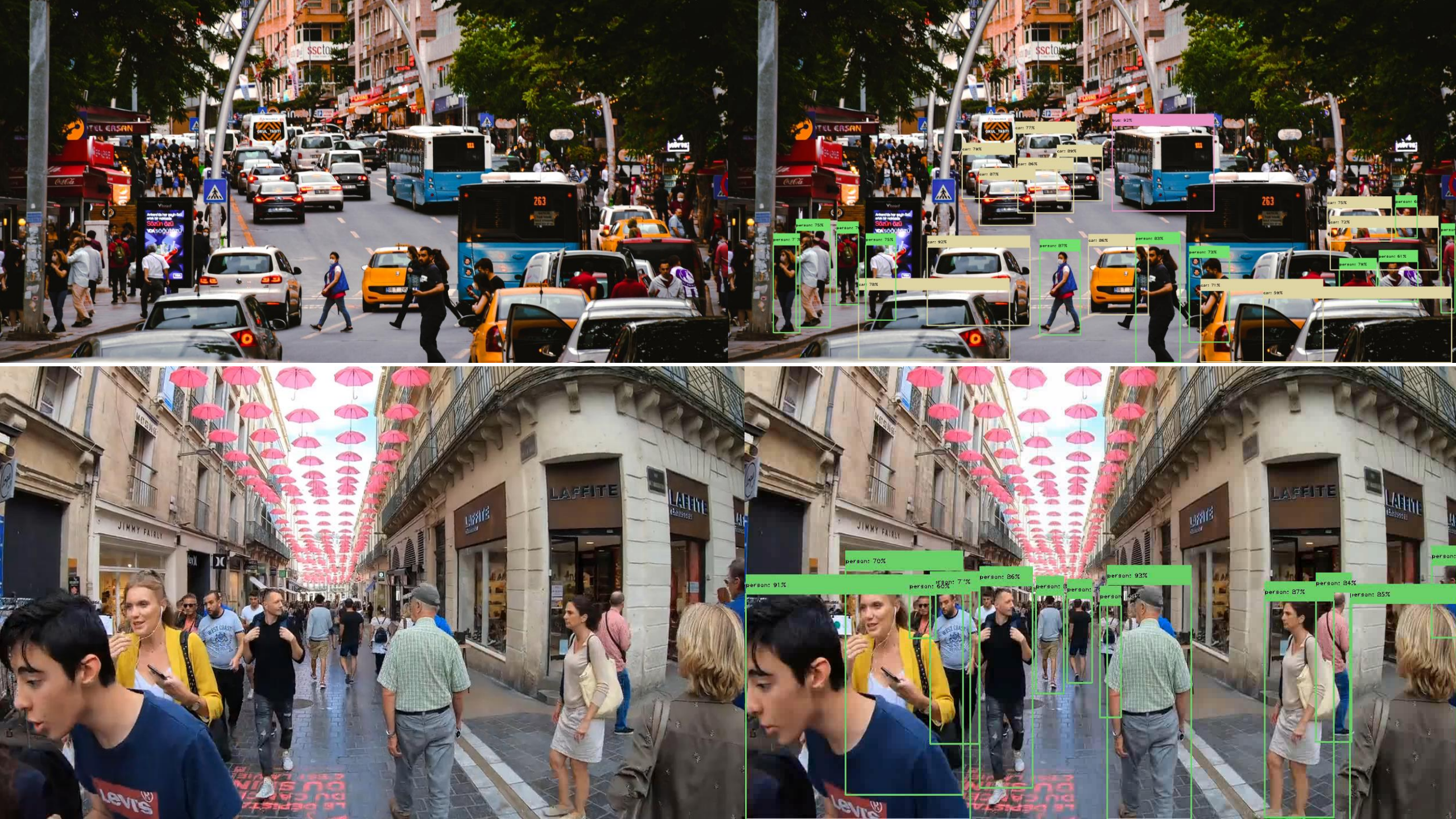
Center X	Center Y	W	H	Confidence	Prob Score class -1	Prob Score class -2	...	Prob Score class – 20
				. . .				

How it begins –




```
0 0.605 0.37733333333333335 0.054 0.22133333333333333
0 0.511 0.388 0.058 0.184
0 0.454 0.38 0.036 0.18933333333333333
0 0.869 0.43466666666666665 0.13 0.512
0 0.149 0.36133333333333334 0.038 0.136
0 0.21 0.3546666666666667 0.036 0.13866666666666666
0 0.29 0.48 0.18 0.544
0 0.568 0.38 0.044 0.21066666666666667
8 0.461 0.5866666666666667 0.318 0.32
19 0.053 0.2866666666666667 0.098 0.14666666666666667
```


```
{
  "version": "5.4.1",
  "flags": {},
  "shapes": [
    {
      "label": "person",
      "points": [
        [
          98.955223880597,
          76.30597014925372
        ],
        [
          184.92537313432837,
          284.0671641791045
        ]
      ],
      "group_id": 0,
      "description": "Person",
      "shape_type": "rectangle",
      "flags": {},
      "mask": null
    },
```


HOTEL ERSAN
263
JIMMY FAIRLY
LAFFITE
Levi's

Week 2 –

YOLOv5 (You Only Look Once) –

Original paper- https://arxiv.org/pdf/1506.02640v5.pdf

More explanation - https://www.datacamp.com/blog/yolo-object-detection-explained

Architecture-

- Input size is 448*448
- Works as a regression task while previous Algorithms used classification to detect objects
- overall 24 convolutional layers, four max-pooling layers, and two fully connected layers
- The algorithm works based on the following four approaches:
  - Residual blocks
  - Bounding box regression
  - Intersection Over Unions or IOU for short
  - Non-Maximum Suppression.

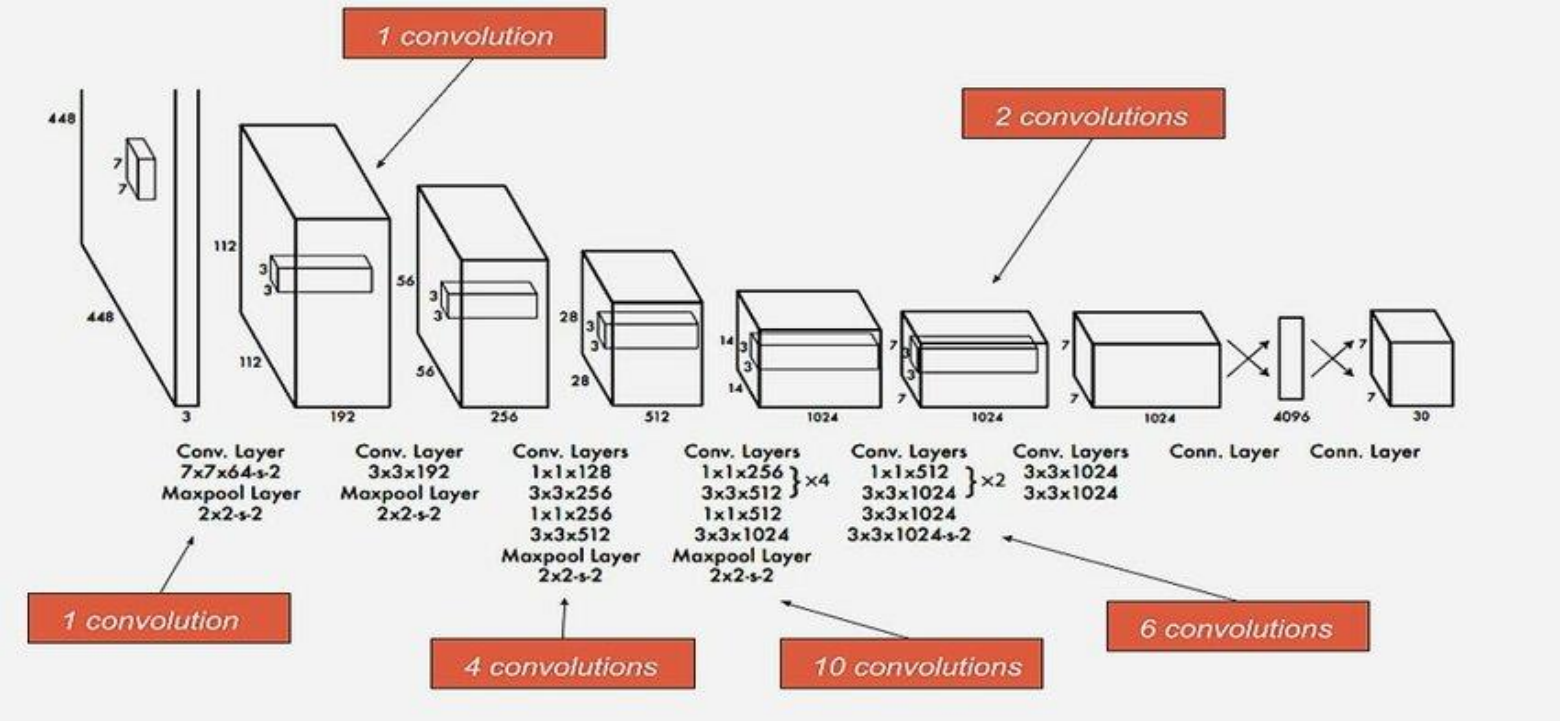
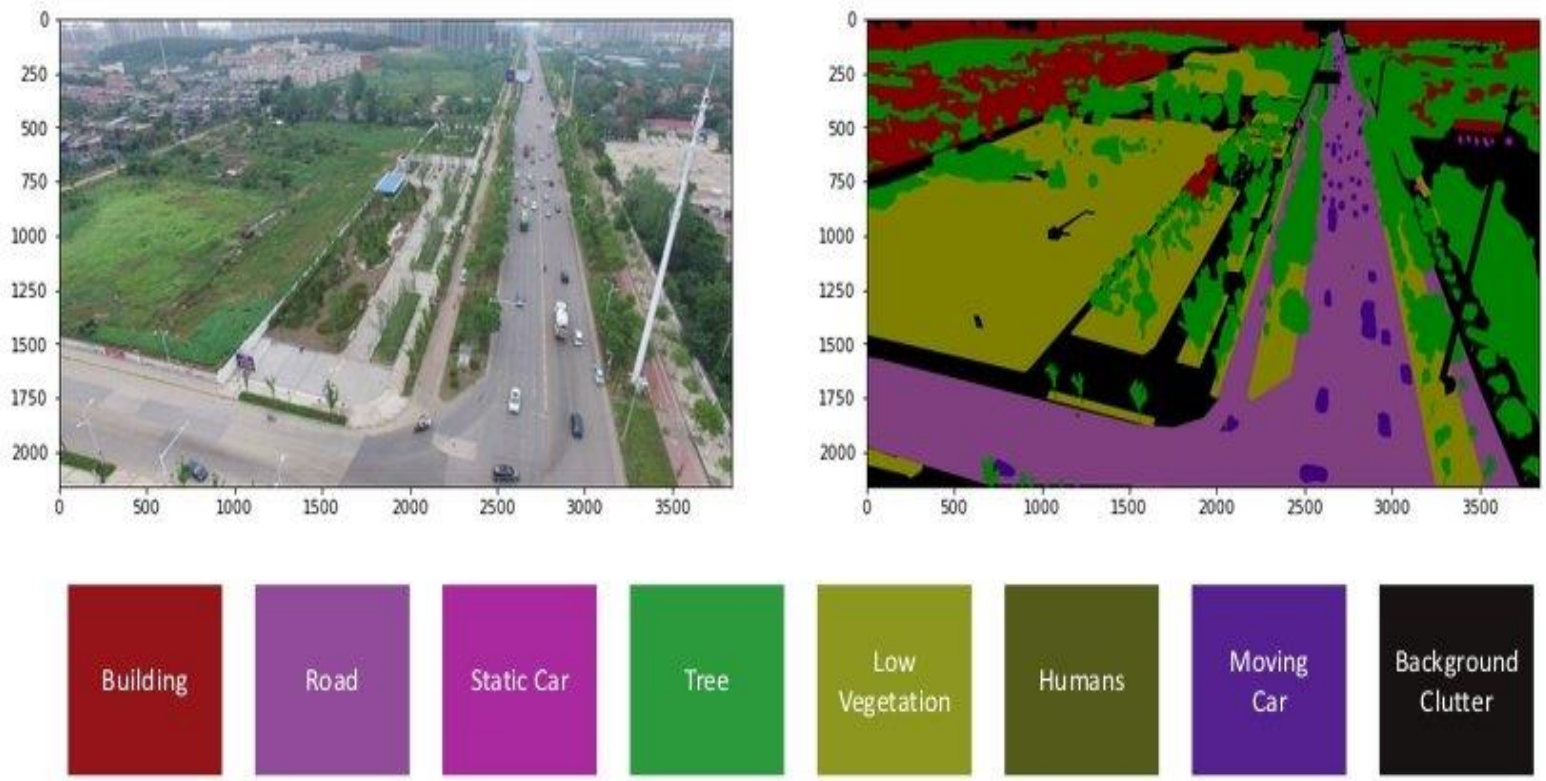
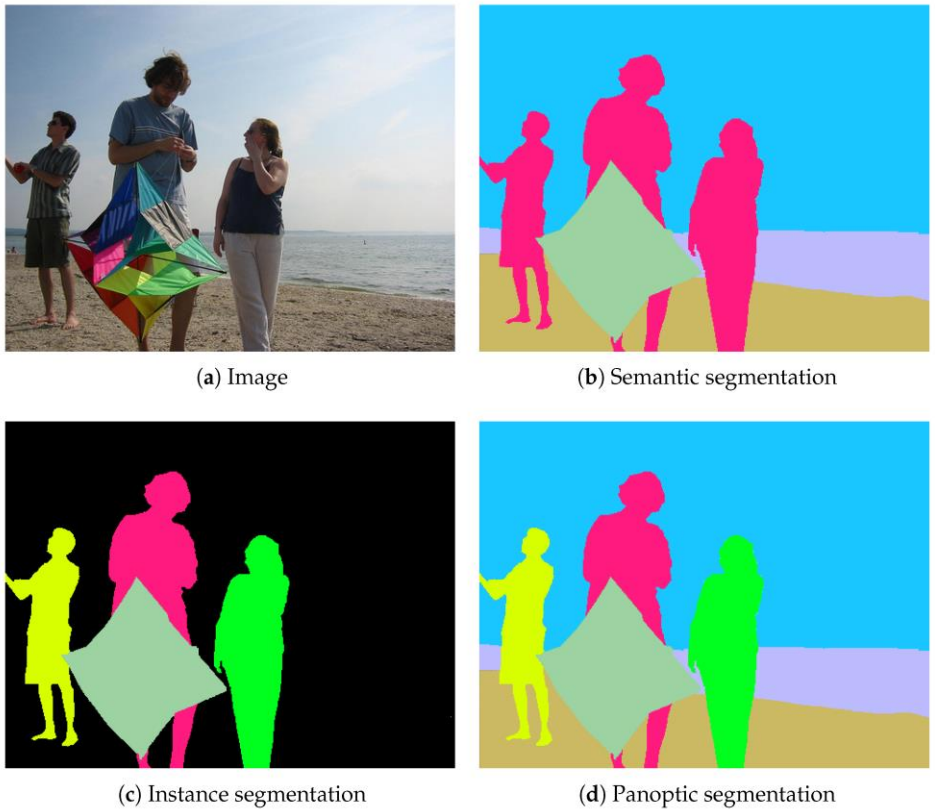


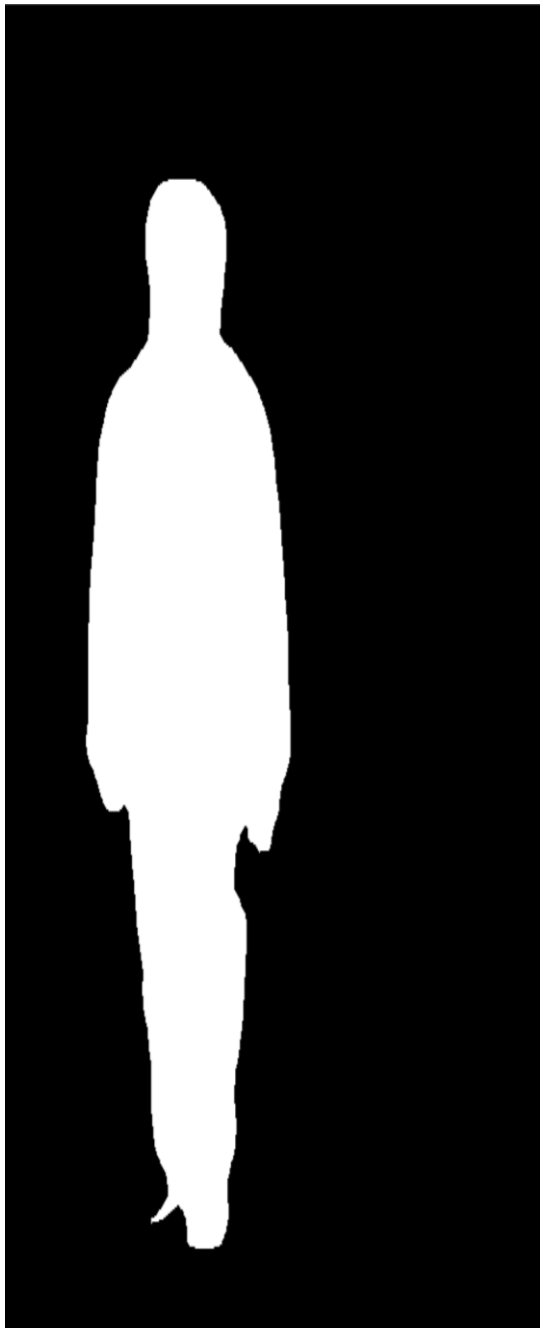
Image Segmentation- Divide an image into multiple parts or regions of interest that belong to the same class.



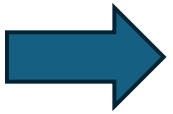


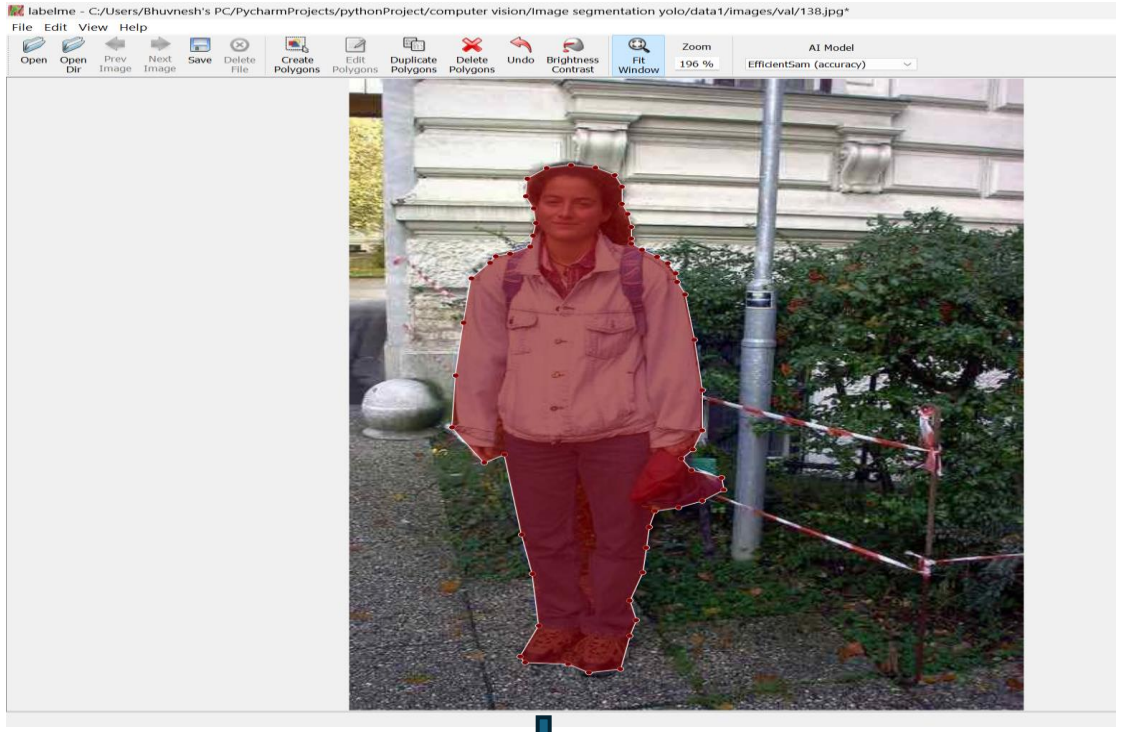
Semantic Segmentation- Differentiates among classes i.e. for every pixel finds its class.
Instance segmentation- Differentiates among ,many instances of, class of interest i.e. for every pixel, the specific belonging instance of the class.
Panoptic segmentation combines both semantic and instance segmentation. Like semantic segmentation, panoptic segmentation is an approach that identifies, for every pixel, the belonging class. Moreover, like in instance segmentation, panoptic segmentation distinguishes different instances of the same class.

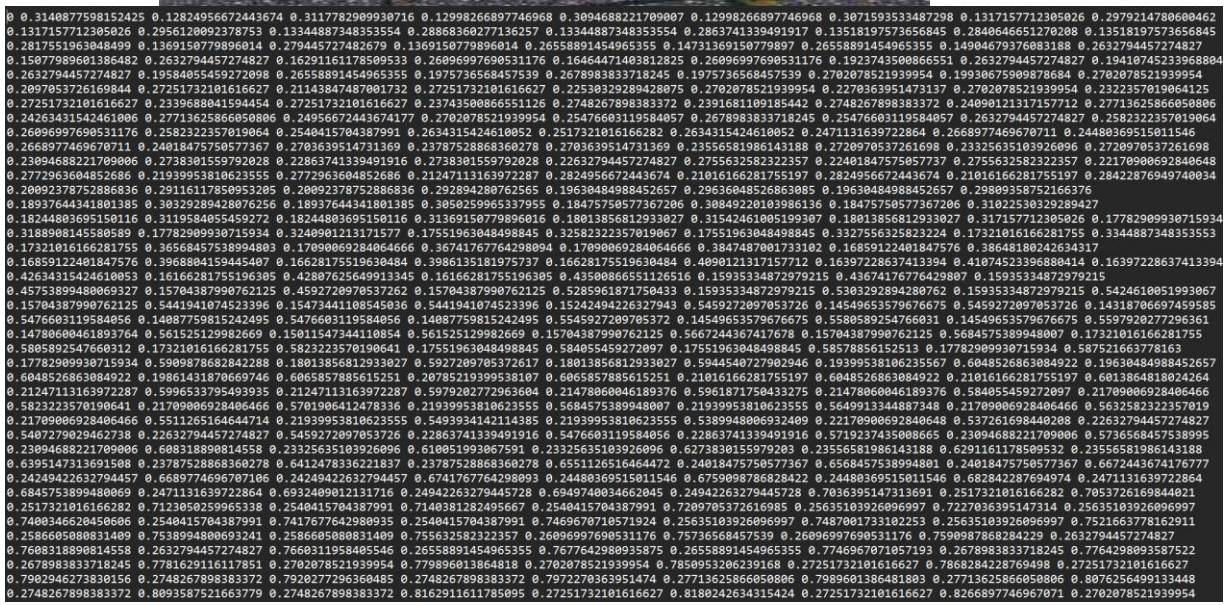
Semantic Segmentation with class of interest as Person

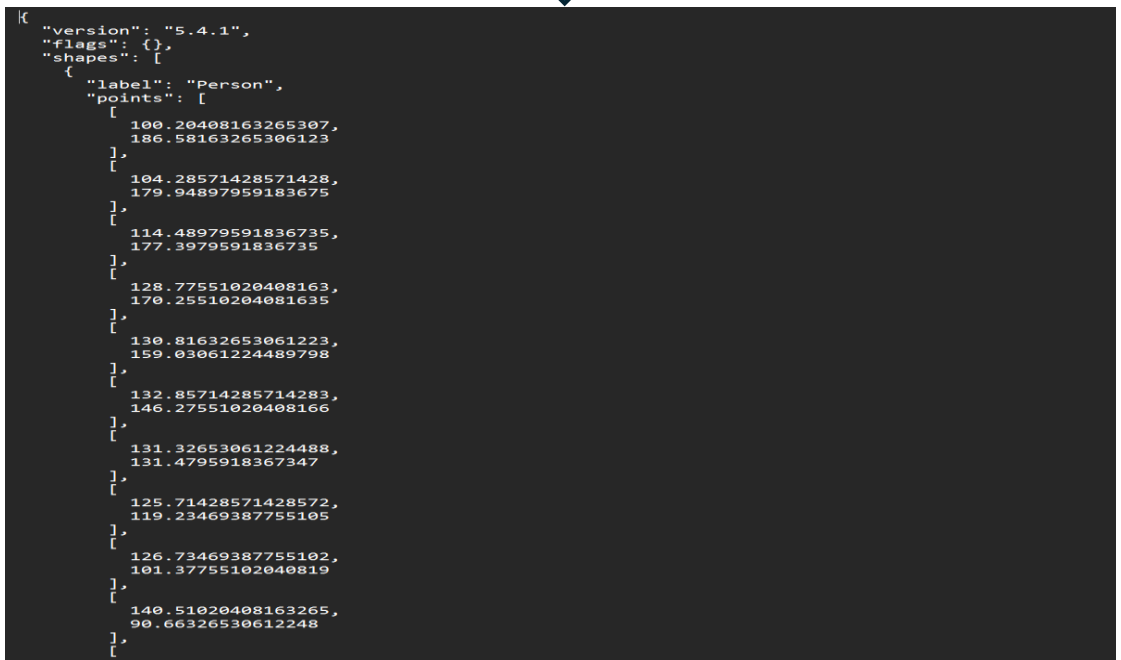


How it begins-









Problem statement- Make a binary mask of a given image, segmenting the person.

Data preparation-

- images are collected from google images.
- Data is divided into train and test.
- Binary masks of the given images are generated using online segmentation tool CVAT.ai
- From binary masks text files ,containing the class and the annotations of the mask ,are extracted
- These text files are the input for the YOLOv8n Model.

Training-

- Make a YAML file that contains nc (number of classes), names (name of classes), train (train data path), val (test data path)
- Install ultralytics module using

  !pip install ultralytics
- Use YOLOv8n Model for the training on custom dataset. (YOLO provides n,s,m,l,x varients of the pretrained model)
- We have exported the last.pt model.

Predictions-

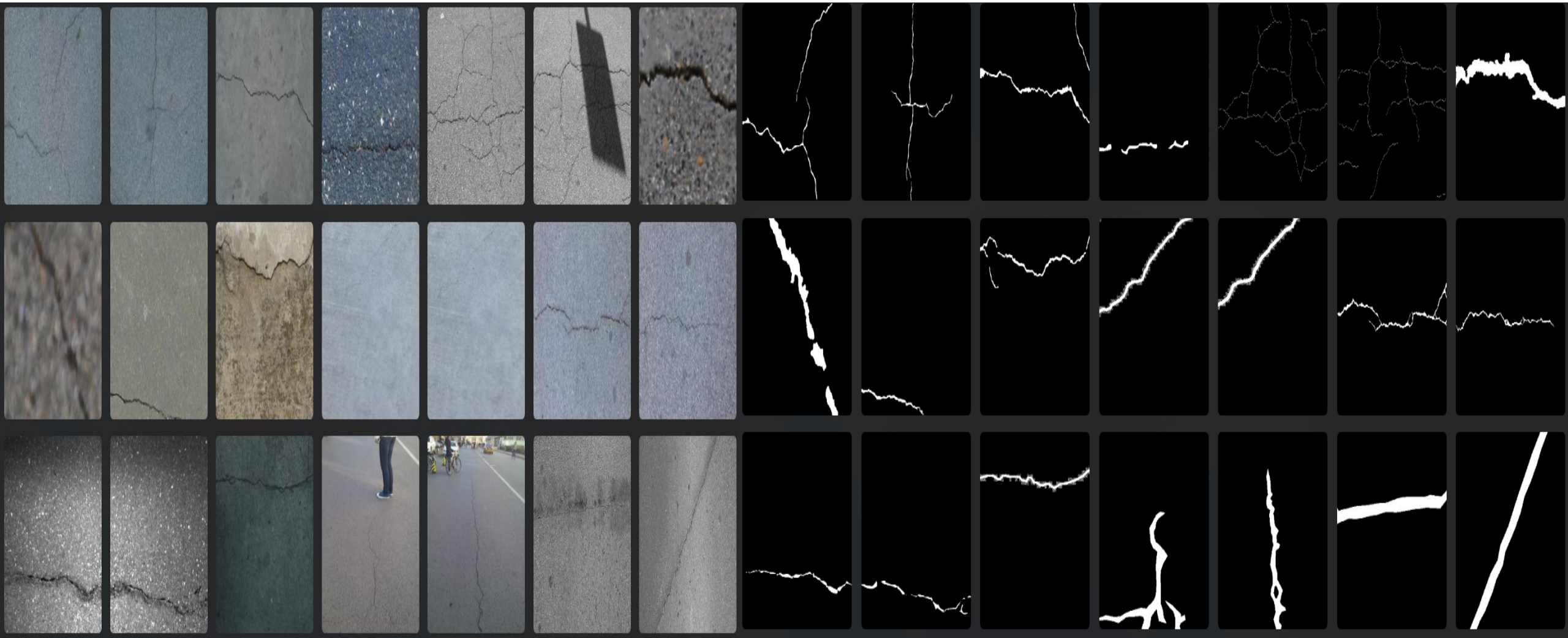
- Write a function that takes a image and convert it into 640*640 image.
- Make a model object using the model which was extracted from training
- Use the model object to predict the modified image.
- The prediction from model will contain the class as 0 and the normalized co-ordinated of the polygons containing the person
- Use this information to make binary masks

Dataset

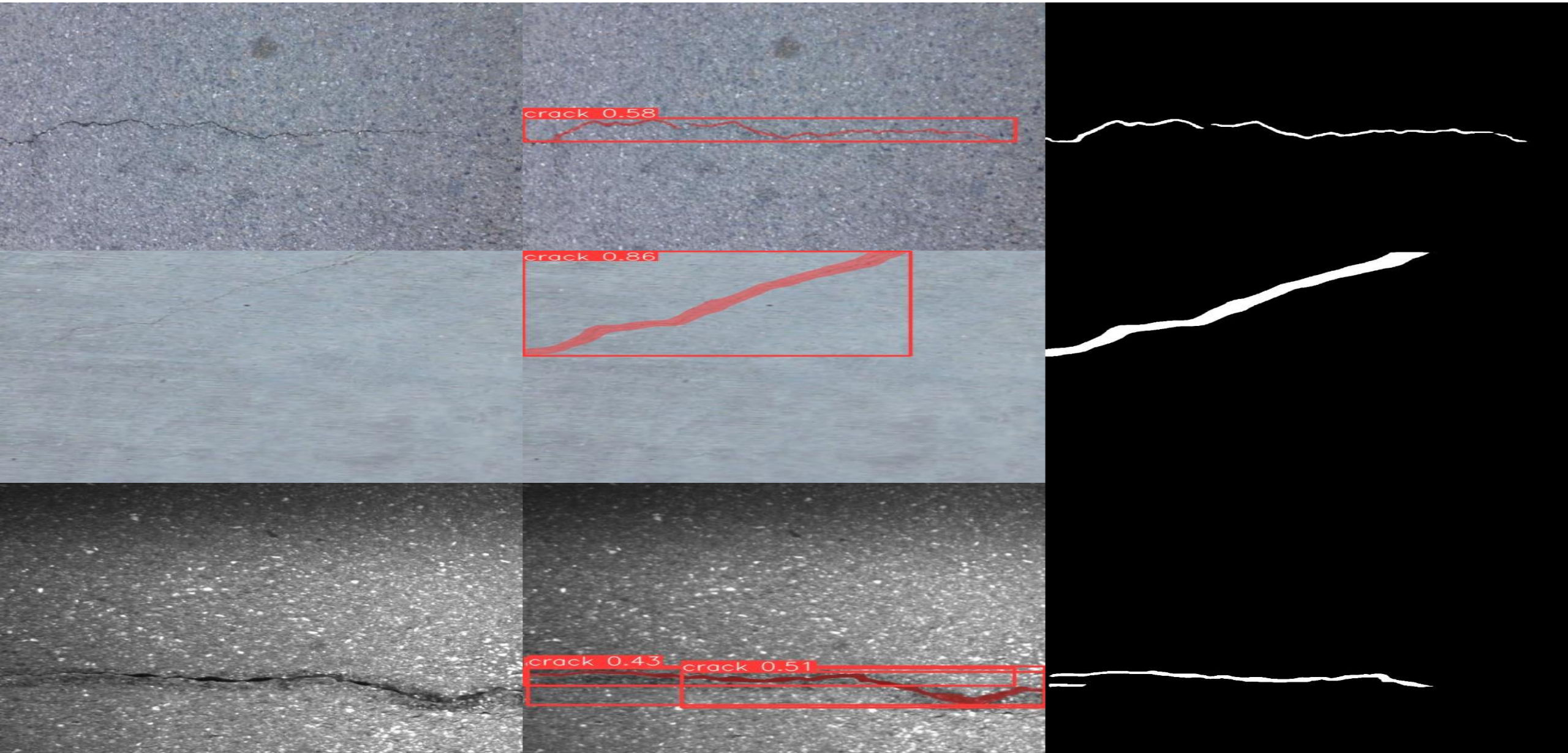
As per our **Main** problem statement, we have to find the cracks in the pictures, We will be Focussing on **Instance segmentation** with class of interest as **Cracks.**

- The Dataset contains various types (Deep, wide, thin, hairline) of cracks on various surfaces (concrete, walls, plaster, roof) of cracks.

Images **Ground Truth**



- Ground Truths are generated using the polygons obtained using LabelMe module.
- The ground Truths are used to train the model (YOLOv8)
- The results -



Sample 4

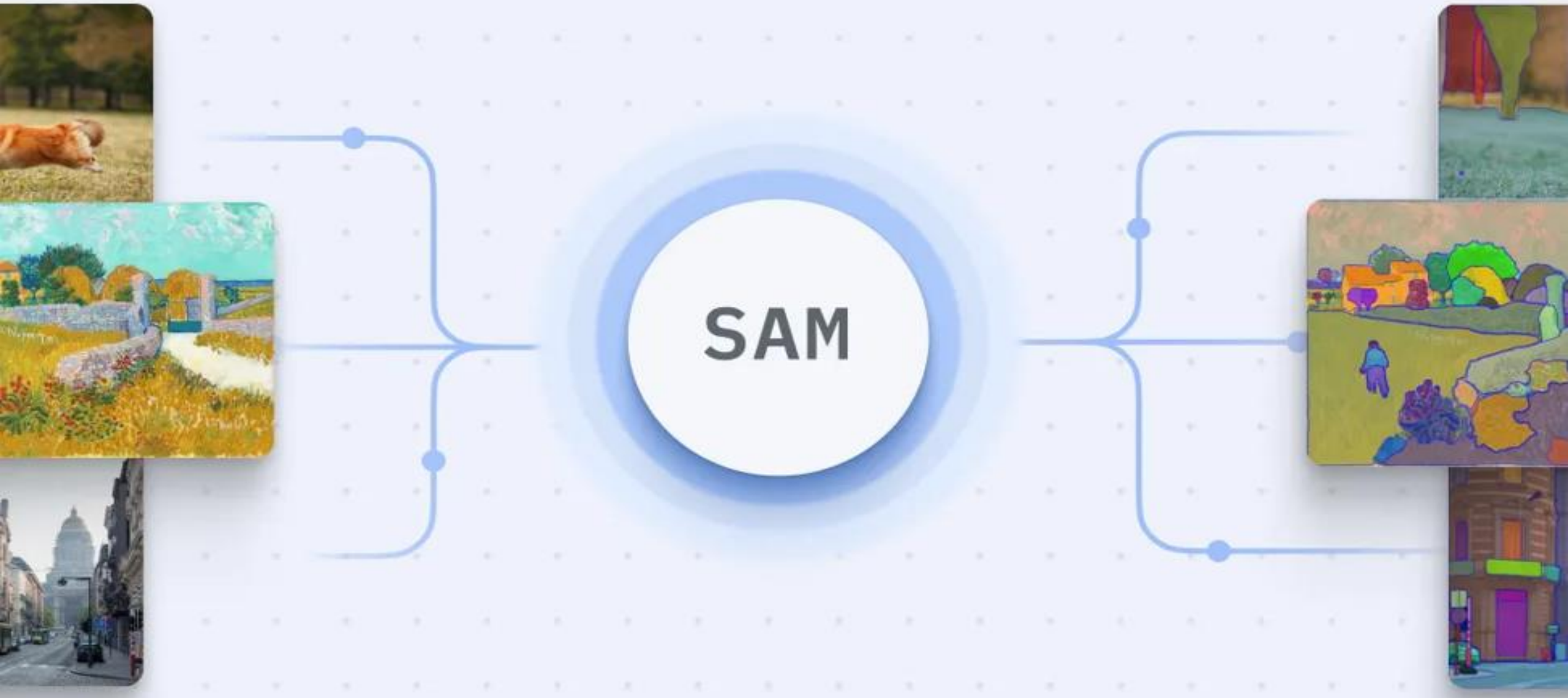
Sample 5

Sample 6

Sample 7

train/seg_loss	1.4363
metrics/precision(M)	0.42349
metrics/recall(M)	0.33333
metrics/mAP50(M)	0.29186
metrics/mAP50-95(M)	0.06245
val/seg_loss	1.8207

Segment Anything Model



Segment Anything Model (SAM)

Segment Anything (SAM) is an image segmentation model developed by Meta AI. This model can identify the precise location of specific objects in an image or every object in an image. SAM was released in April 2023.
SAM is open source, released under an Apache 2.0 license.

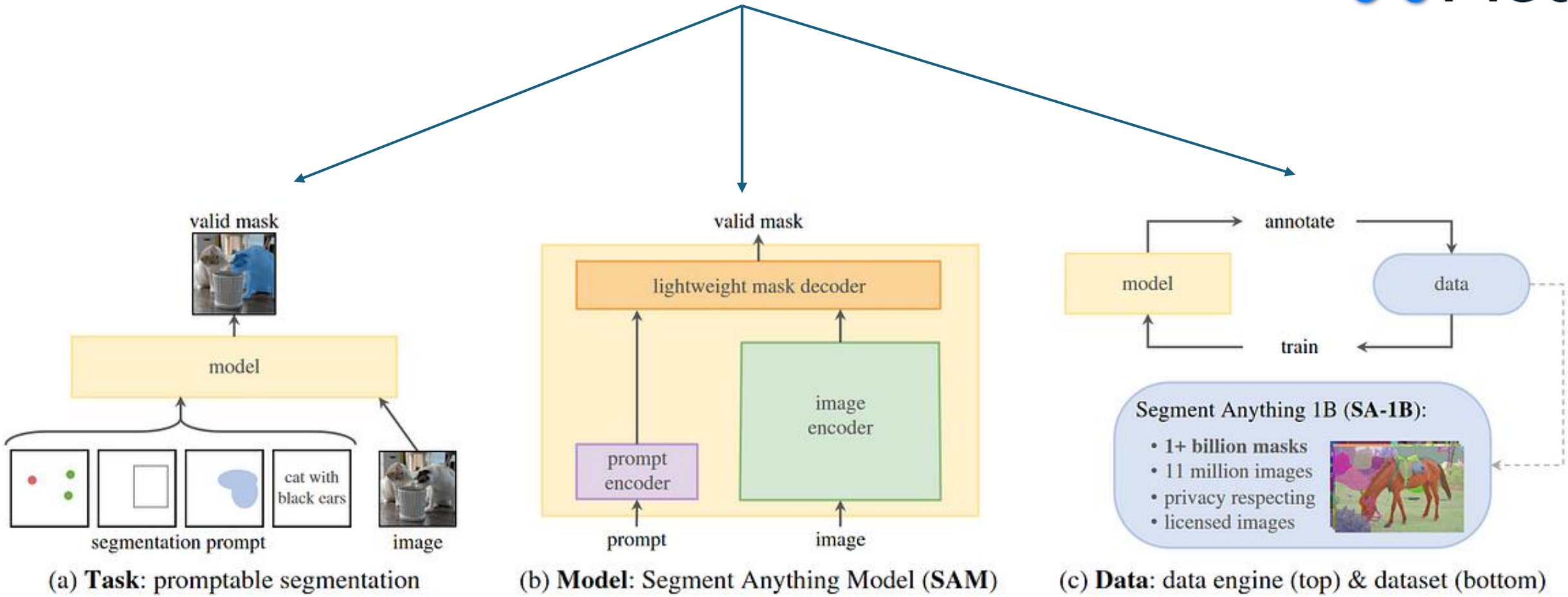
Segment Anything introduces three innovations:

1. Promptable Segmentation Task:
- Points (for example points selected by the user by mouse click)
- Boxes (a rectangle defined by the user)
- Text prompt (for example "find all the dogs in this image")
- A combination of the above (for example a box and a background point)

2. Segment Anything Model
- A fast encoder-decoder model (takes 50ms in a web browser to generate a mask given a prompt)
- Ambiguity-aware: for example given a point, it may correspond to multiple masks and the model should return the three most likely (the part, the subpart and the whole).

3. Segment Anything Dataset (and its Segment Anything Engine)
- 1.1 billion segmentation masks collected with the Segment Anything Engine
- All the masks have been generated automatically! (no human supervision!)

Segment Anything Model (SAM)

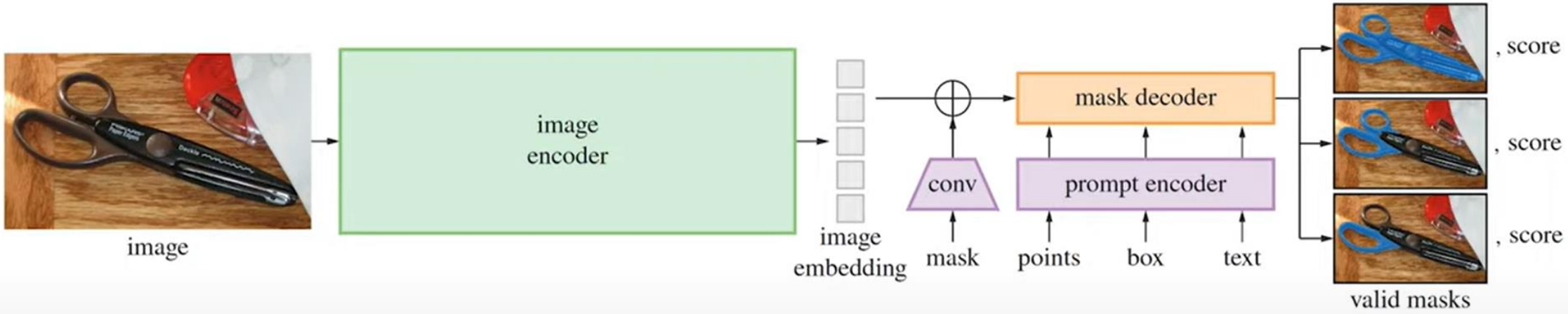


(a) **Task**: promptable segmentation

(b) **Model**: Segment Anything Model (**SAM**)

(c) **Data**: data engine (top) & dataset (bottom)

Working of SAM model-



A heavyweight image encoder outputs an image embedding that can then be efficiently queried by a variety of input prompts to produce object masks at amortized real-time speed. For ambiguous prompts corresponding to more than one object, SAM can output multiple valid masks and associated confidence scores,

- SAM works by first encoding the image into a high-dimensional vector representation. The image encoder is a vision transformer (ViT-H) model, that has been pre-trained on a massive dataset of images.

- The prompt encoder is a simple text encoder that converts the input prompt into a separate vector representation.

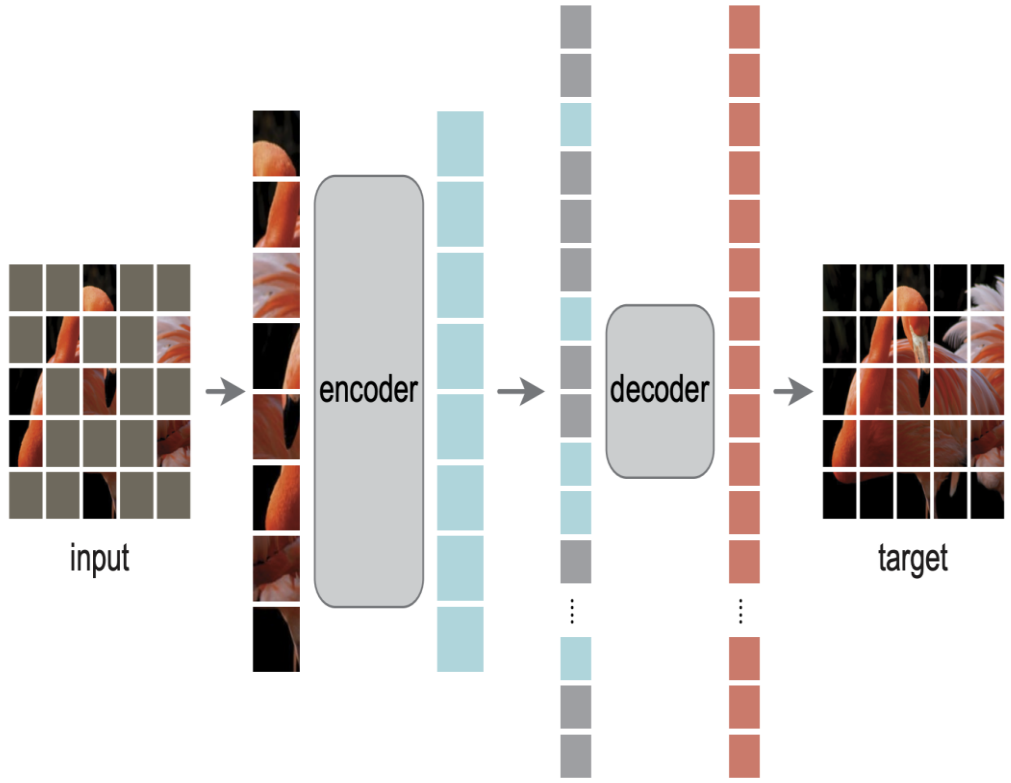
- The two vector representations are then combined and passed to a mask decoder , The mask decoder is a lightweight transformer model that predicts the object mask from the image and prompt embeddings

Prompt Encoder

We consider two sets of prompts: sparse (points, boxes, text) and dense (masks). We represent points and boxes by positional encodings [95] summed with learned embeddings for each prompt type and free-form text with an off-the-shelf text encoder from CLIP [82]. Dense prompts (i.e., masks) are embedded using convolutions and summed element-wise with the image embedding.

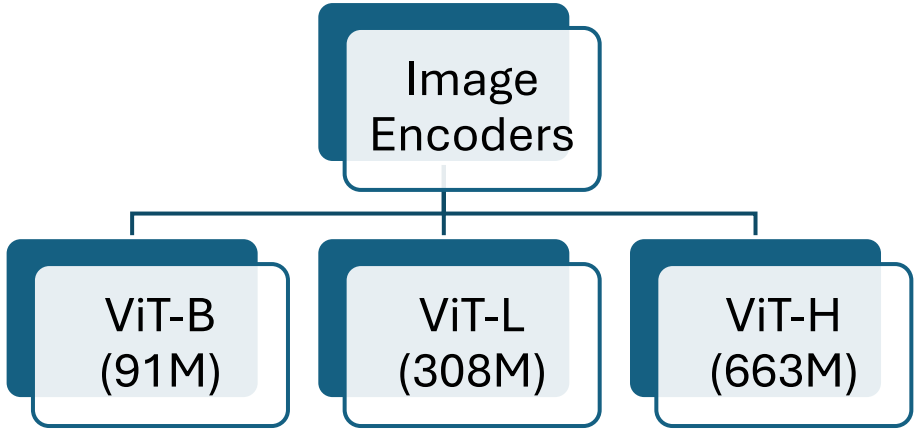
Image Encodere

This is Motivated by scalability and powerful pre-training methods, we use an MAE(Masked Auto Encoder) pre-trained Vision Transformer (ViT) [33] minimally adapted to process high resolution inputs [62]. The image encoder runs once per image and can be applied prior to prompting the model.



Types

- The SAM model can be loaded with 3 different encoders:



Training

- Provide image and a prompt, The prompt can be (Text, Box, including points)
- Average Training time 2.15 sec / iter for vit large
- Average Training time for vit base model is 1 iteration per second

```
EPOCH: 1
Mean loss: 5.490579358921495
  0%|          | 1/215 [00:00<02:10,  1.64it/s] loss: 4.487935543060303
  1%|          | 2/215 [00:01<02:09,  1.65it/s] loss: 2.514836072921753
  1%|          | 3/215 [00:01<02:08,  1.65it/s] loss: 4.278687477111816
  2%|          | 4/215 [00:02<02:07,  1.66it/s] loss: 3.3740503787994385
  2%|          | 5/215 [00:03<02:06,  1.66it/s] loss: 2.001035213470459
  3%|          | 6/215 [00:03<02:05,  1.66it/s] loss: 3.29148530960083
  3%|          | 7/215 [00:04<02:04,  1.67it/s] loss: 3.7787740230560303
  4%|          | 8/215 [00:04<02:04,  1.67it/s] loss: 3.4222958087921143
  4%|          | 9/215 [00:05<02:03,  1.67it/s] loss: 4.409018516540527
  5%|          | 10/215 [00:06<02:04,  1.64it/s] loss: 1.745933175086975
  5%|          | 11/215 [00:06<02:03,  1.65it/s] loss: 3.4013991355895996
  6%|          | 12/215 [00:07<02:02,  1.65it/s] loss: 4.767216205596924
  6%|          | 13/215 [00:07<02:02,  1.65it/s] loss: 2.1399831771850586
  7%|          | 14/215 [00:08<02:01,  1.65it/s] loss: 5.946110725402832
  7%|          | 15/215 [00:09<02:00,  1.66it/s] loss: 4.916722774505615
  7%|          | 16/215 [00:09<01:59,  1.66it/s] loss: 4.766115665435791
  8%|          | 17/215 [00:10<01:59,  1.66it/s] loss: 2.1327152252197266
  8%|          | 18/215 [00:10<01:58,  1.66it/s] loss: 4.637602806091309
  9%|          | 19/215 [00:11<01:58,  1.66it/s] loss: 4.795048713684082
  9%|          | 20/215 [00:12<01:57,  1.65it/s] loss: 3.1774535179138184
 10%|          | 21/215 [00:12<01:56,  1.66it/s] loss: 4.448383808135986
 10%|          | 22/215 [00:13<01:56,  1.66it/s] loss: 3.338754653930664
```

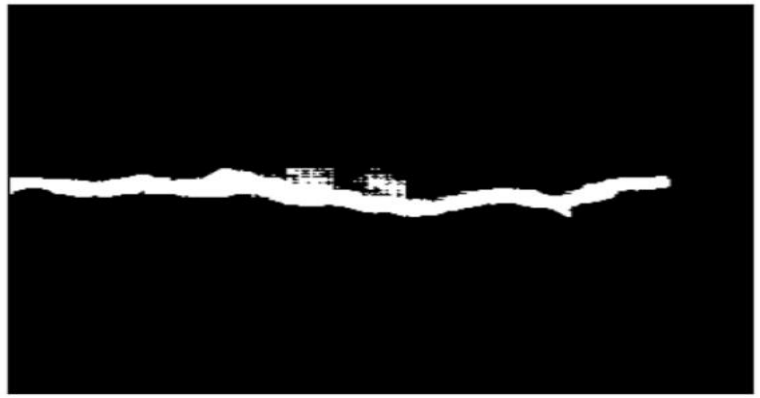
Prediction

Prediction how will you provide a prompt-

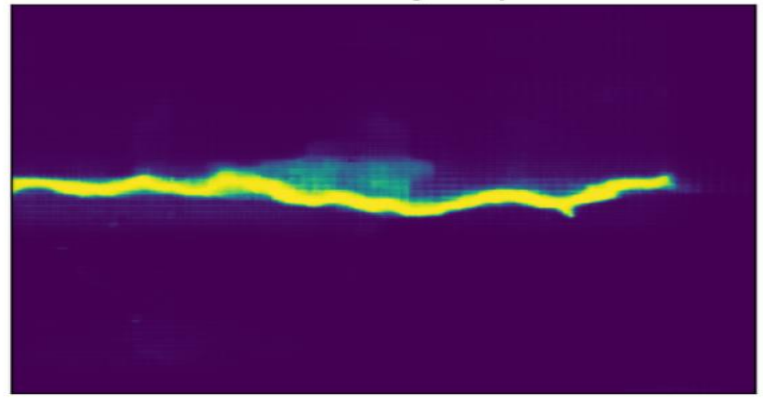
- Provide predefined bounding boxes of fix size
- Use a model that can provide these bounding boxes (YOLO)

Image

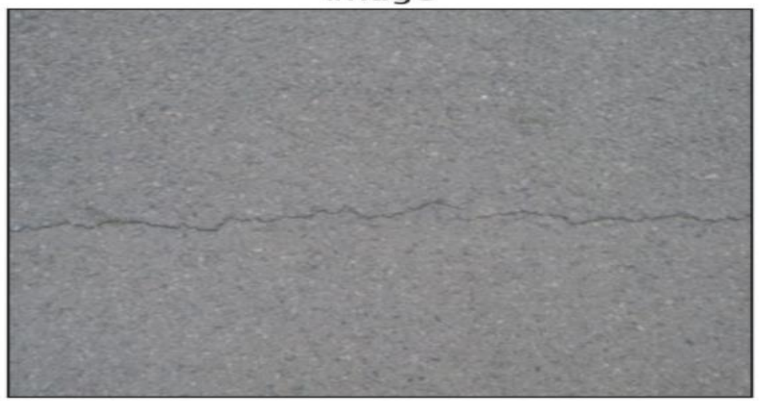
Mask



Probability Map

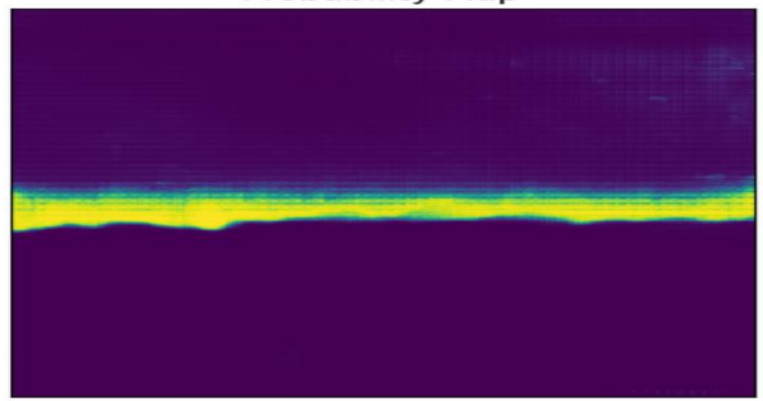


Image



Mask

Probability Map

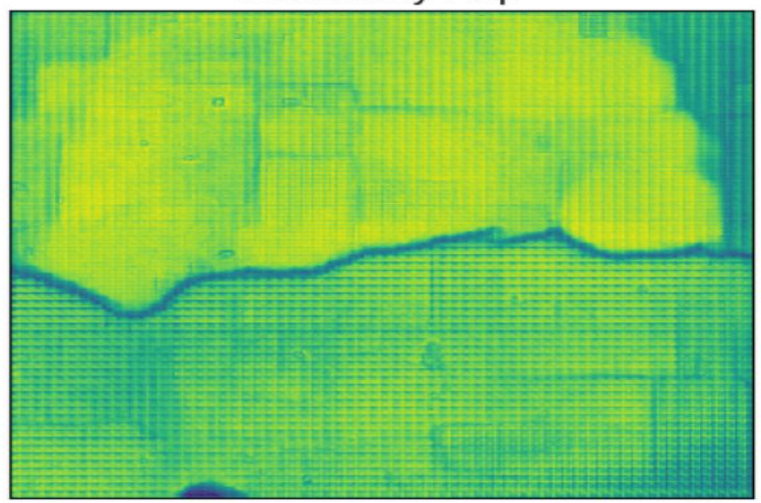


Image

Image

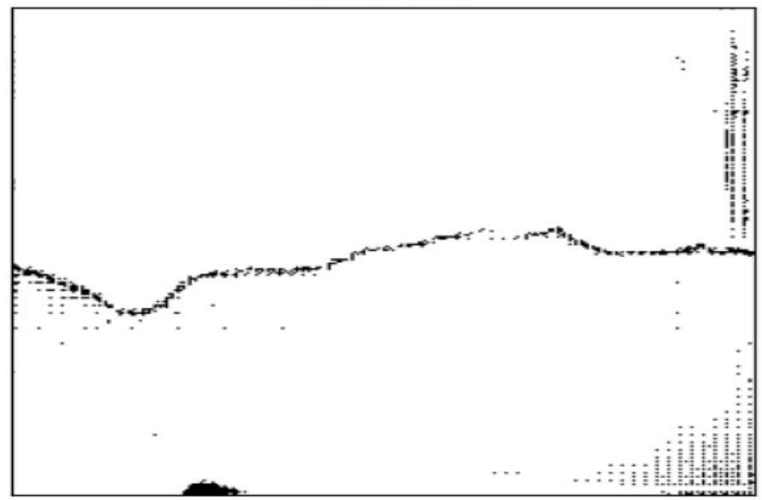
Mask

Probability Map



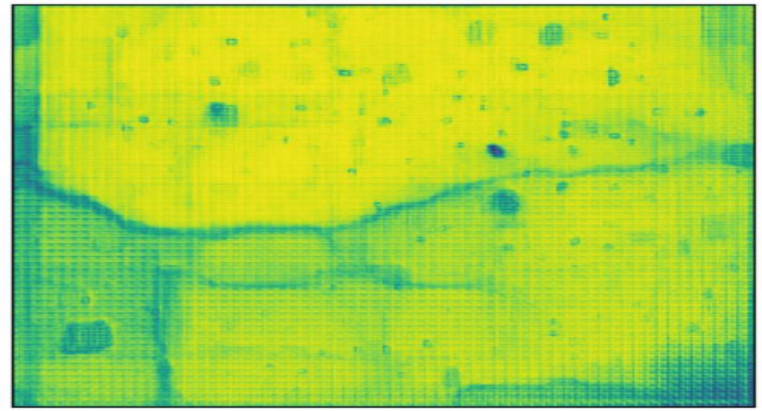
Probability Map

Prediction

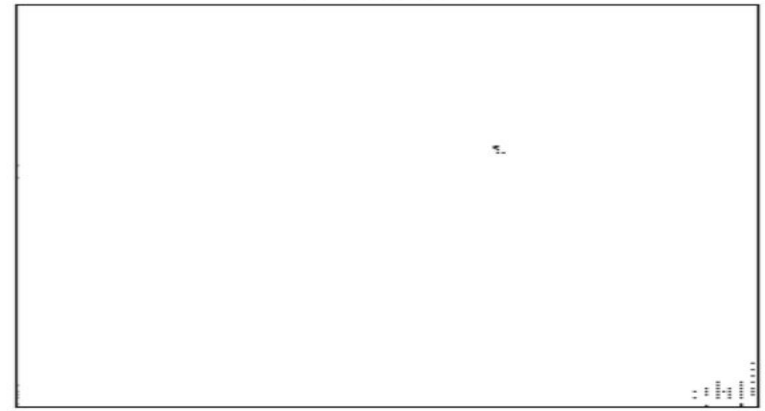


Image

Probability Map

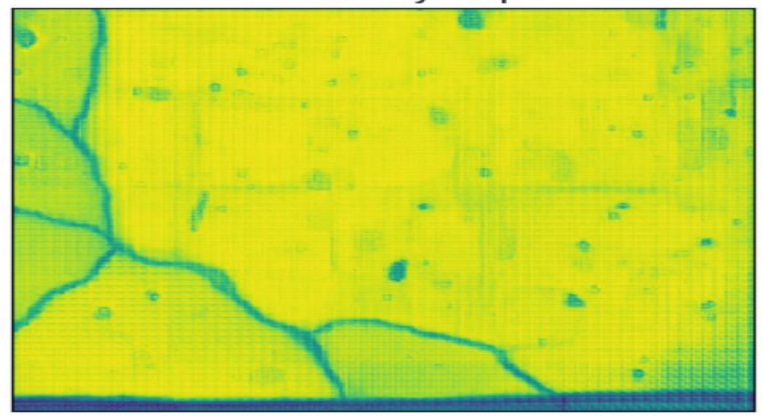


Prediction

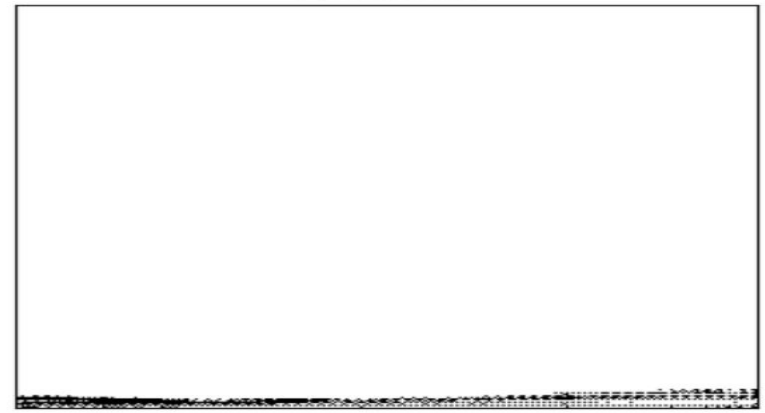


Image

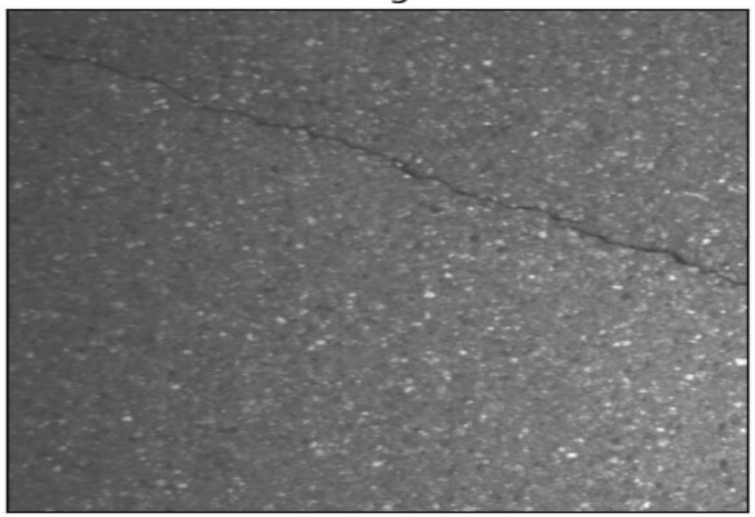
Probability Map



Prediction

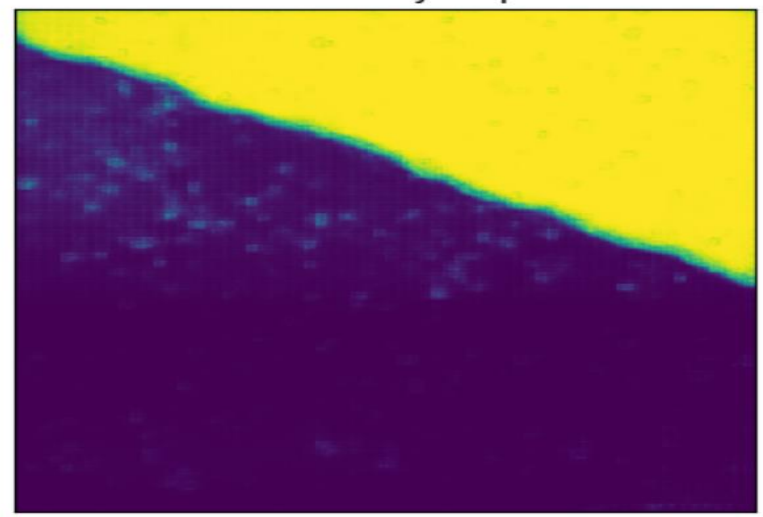


Image



Mask

Probability Map



Meta

Detectron2

Real-Time Crack detection and Segmentation

Detectron2

What is detectron2 ?

Detectron2 is a robust and flexible object detection library developed by Facebook AI Research. It is built on PyTorch.

How does Detectron2 work?

Detectron2 uses a two-stage approach to object detection:

1. It uses a region proposal network (RPN) to generate a set of candidate regions.
2. Then uses Mask R-CNN model to classify and segment the candidate regions.(for segmentation)

Region Proposal Network (RPN)

- RPN is a convolutional neural network that takes an image as input and generates a set of candidate regions.
- The RPN is trained to predict a score for each region, indicating how likely it is to contain an object.
- It also predicts a bounding box for each region.
- Mask R-CNN model predicts a segmentation mask for each region.

Detectron2 in itself is not a architecture, instead it provides us baselines based on **3 different backbone combinations**:

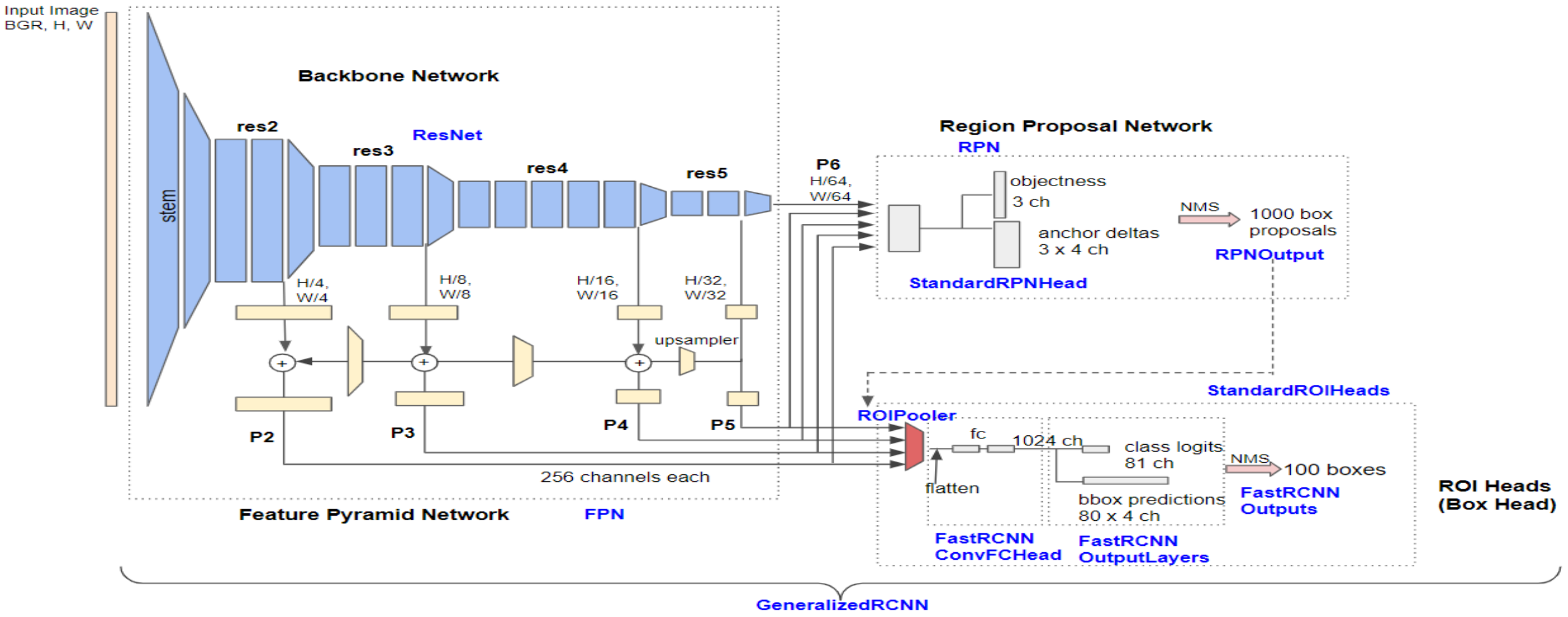
- **FPN**: Use a ResNet+FPN backbone with standard conv and FC heads for mask and box prediction, respectively. It obtains the best speed/accuracy tradeoff, but the other two are still useful for research.

- **C4**: Use a ResNet conv4 backbone with conv5 head. The original baseline in the Faster R-CNN paper.

- **DC5** (Dilated-C5): Use a ResNet conv5 backbone with dilations in conv5, and standard conv and FC heads for mask and box prediction, respectively. This is used by the Deformable ConvNet paper.

We can find combinations of Pre-Trained Models in Detectron 2-

- It has provided baselines for R-CNN, FASTER R-CNN, RetinaNet, Mask R-CNN Models.
- And the Models are trained on COCO , LVIS, PASCAL VOC, CityScapes Datasets.
- Different combinations have differences in Mask AP, Box AP, Inference Time, epochs and model size.
- One can try any model for specific purpose.
- All COCO models were trained on train2017 and evaluated on val2017



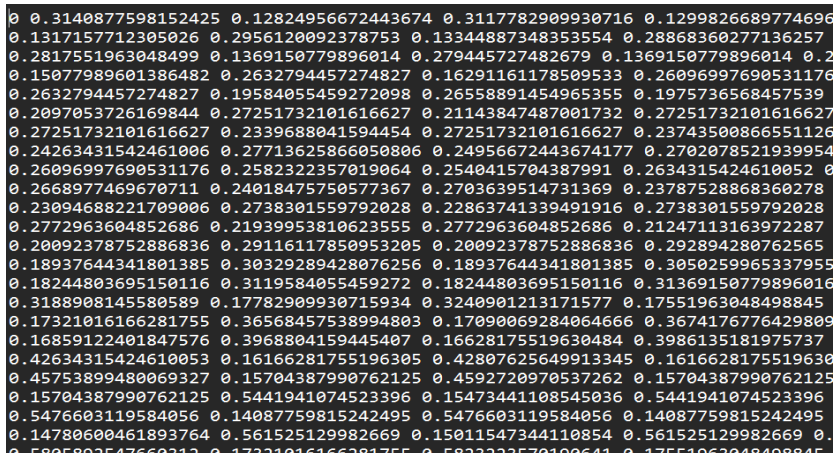
Detectron2 Model Zoo

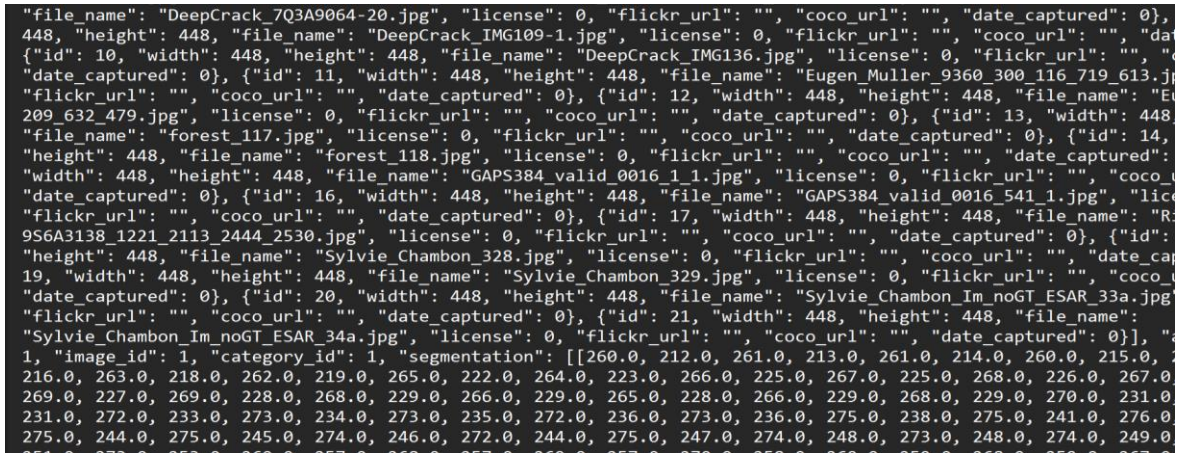
COCO Instance Segmentation Baselines with Mask R-CNN

Name	lr sched	train time (s/iter)	inference time (s/im)	train mem (GB)	box AP	mask AP	model id	download
R50-C4	1x	0.584	0.110	5.2	36.8	32.2	137259246	model | metrics
R50-DC5	1x	0.471	0.076	6.5	38.3	34.2	137260150	model | metrics
R50-FPN	1x	0.261	0.043	3.4	38.6	35.2	137260431	model | metrics
R50-C4	3x	0.575	0.111	5.2	39.8	34.4	137849525	model | metrics
R50-DC5	3x	0.470	0.076	6.5	40.0	35.9	137849551	model | metrics
R50-FPN	3x	0.261	0.043	3.4	41.0	37.2	137849600	model | metrics
R101-C4	3x	0.652	0.145	6.3	42.6	36.7	138363239	model | metrics
R101-DC5	3x	0.545	0.092	7.6	41.9	37.3	138363294	model | metrics
R101-FPN	3x	0.340	0.056	4.6	42.9	38.6	138205316	model | metrics
X101-FPN	3x	0.690	0.103	7.2	44.3	39.5	139653917	model | metrics

Preparing Dataset-

- Input as COCO format.
- Provide 2 folders train, val , each containing images as well as a JSON file.
- Converted annotations from YOLO format to COCO format.





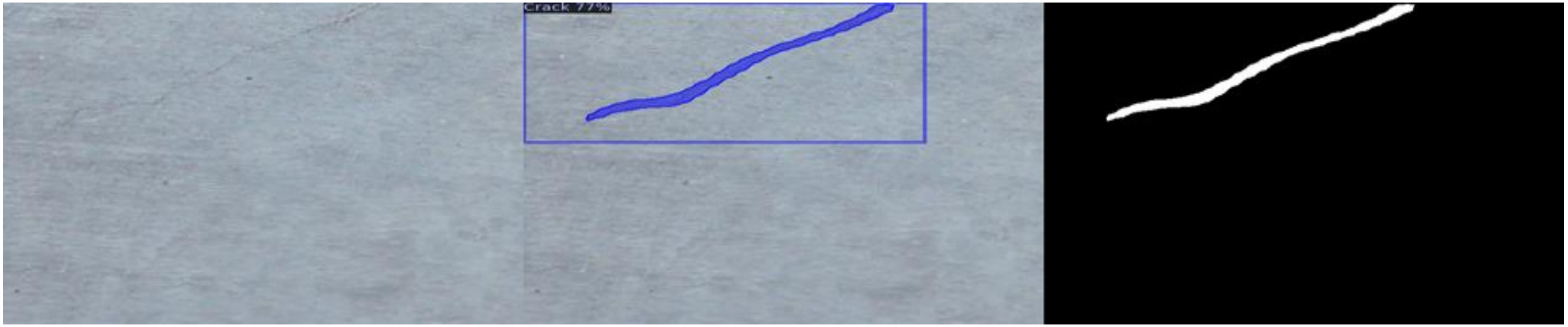
Training -

- Installing Detectron2 using https://github.com/facebookresearch/detectron2/tree/main/detectron2.
- Using a pretrained model “COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml”.
- Registering the data.

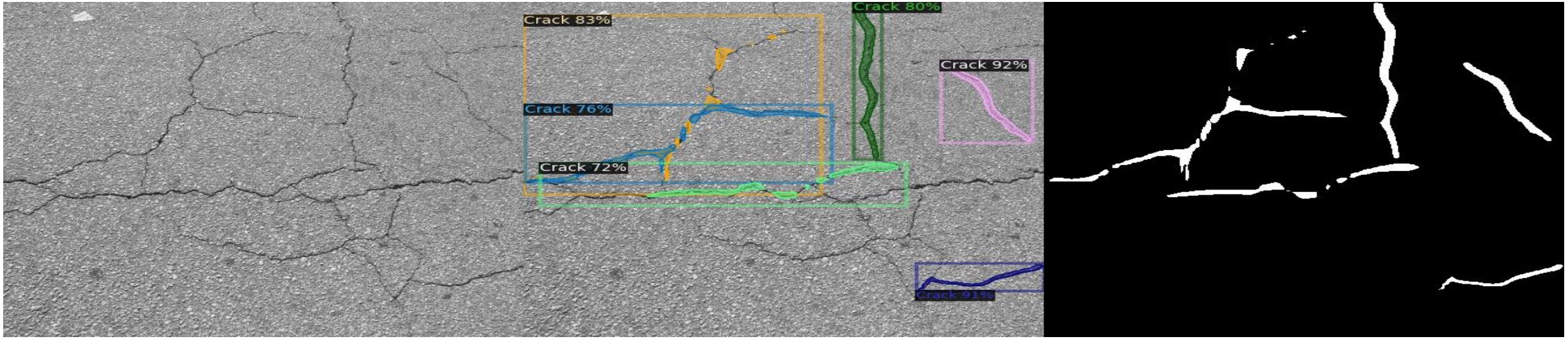
```
[06/19 18:41:48 d2.utils.events]:  eta: 0:00:46  iter: 219  total_loss: 0.5793  loss_cls: 0.1275  loss_box_reg: 0.2176  loss_mask: 0.193  loss_r
/content/detectron2/detectron2/layers/wrappers.py:142: UserWarning: Plan failed with a cudnnException: CUDNN_BACKEND_EXECUTION_PLAN_DESCRIPTOR:
  x = F.conv2d(
/content/detectron2/detectron2/layers/wrappers.py:142: UserWarning: Plan failed with a cudnnException: CUDNN_BACKEND_EXECUTION_PLAN_DESCRIPTOR:
  x = F.conv2d(
/content/detectron2/detectron2/layers/wrappers.py:142: UserWarning: Plan failed with a cudnnException: CUDNN_BACKEND_EXECUTION_PLAN_DESCRIPTOR:
  x = F.conv2d(
[06/19 18:42:17 d2.utils.events]:  eta: 0:00:15  iter: 239  total_loss: 0.5676  loss_cls: 0.1215  loss_box_reg: 0.1747  loss_mask: 0.1935  loss_
/content/detectron2/detectron2/layers/wrappers.py:142: UserWarning: Plan failed with a cudnnException: CUDNN_BACKEND_EXECUTION_PLAN_DESCRIPTOR:
  x = F.conv2d(
[06/19 18:42:34 d2.utils.events]:  eta: 0:00:00  iter: 249  total_loss: 0.5463  loss_cls: 0.1186  loss_box_reg: 0.1756  loss_mask: 0.1951  loss_
[06/19 18:42:34 d2.engine.hooks]: Overall training speed: 248 iterations in 0:06:40 (1.6154 s / it)
[06/19 18:42:34 d2.engine.hooks]: Total training time: 0:06:44 (0:00:04 on hooks)
```

Results-

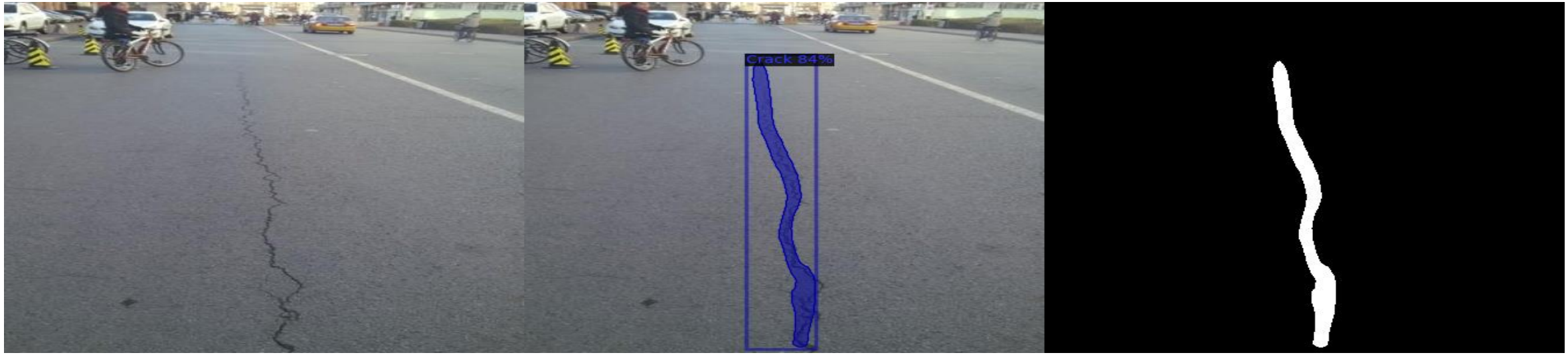
Sample 1



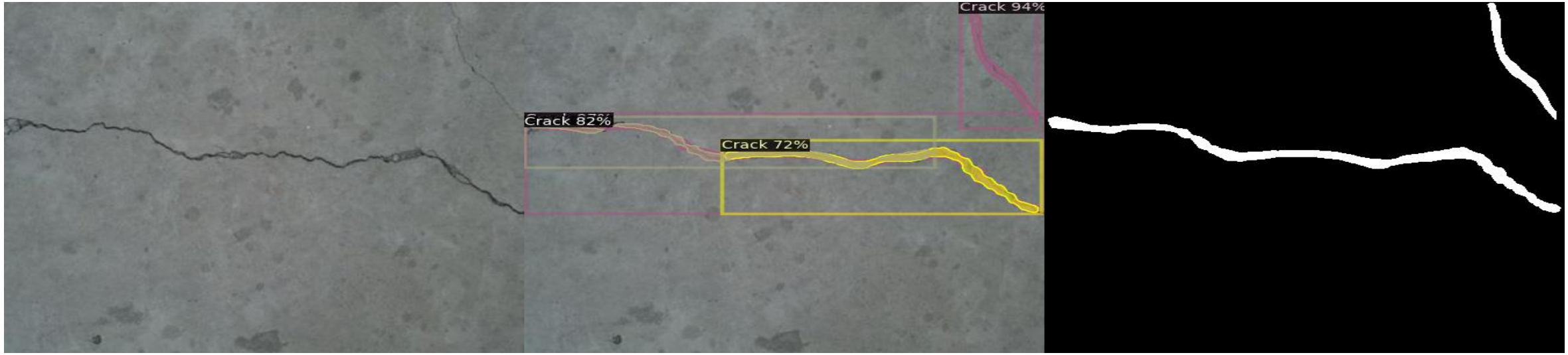
Sample 2



Sample 1



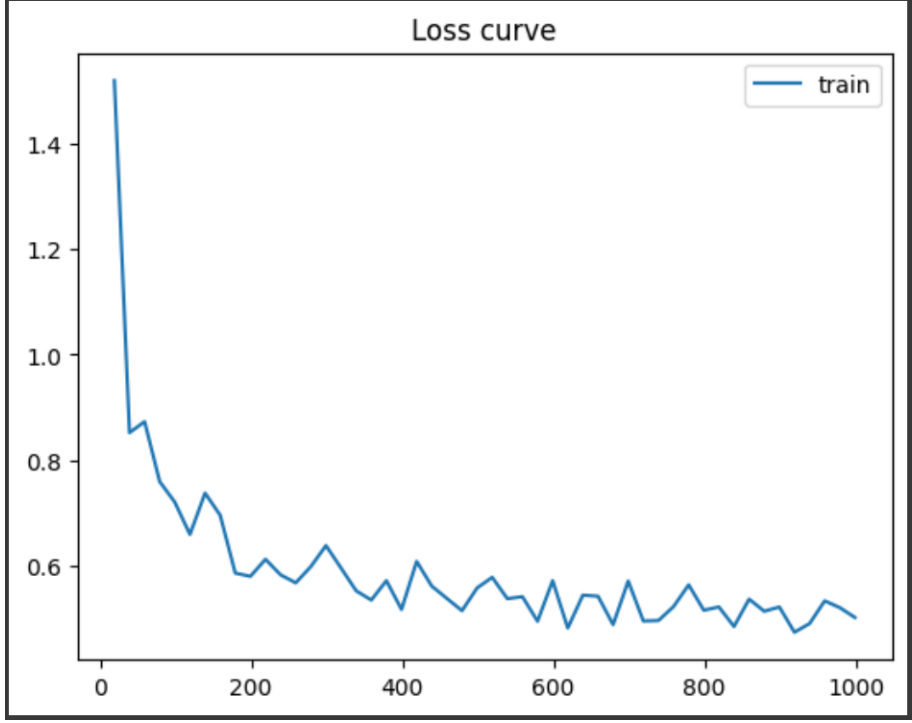
Sample 2



Results-

- We can see that after 600 epochs the loss is stable
- Segmentation loss 0.168
- Evaluation results for segm:

| AP | AP50 | AP75 | APs | APm | APl |
0.785 | 3.804 | 0.083 | 0.000 | 0.859 | 7.723



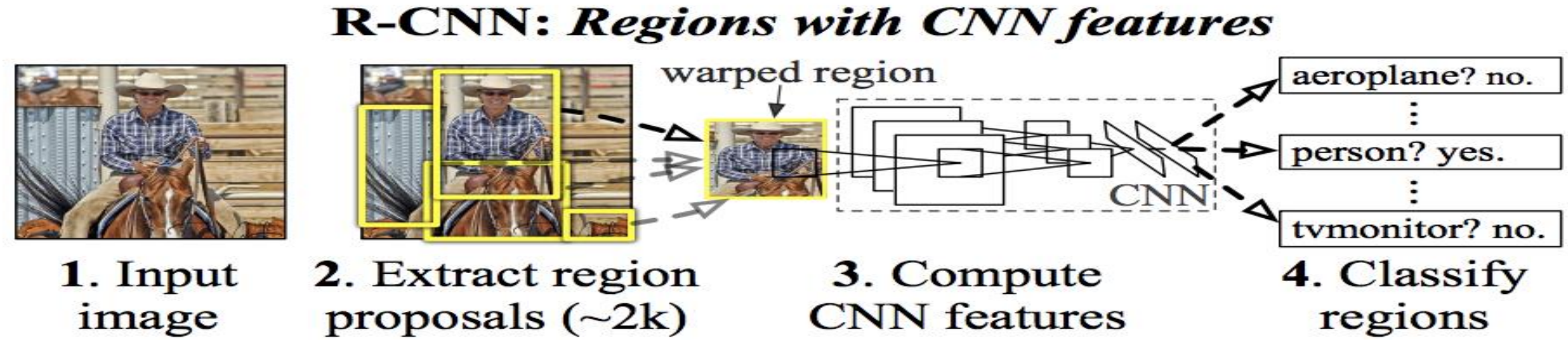
R-CNN

Selective search process to extract 2000 regions from the image (region proposals)
Step 1: Generate many candidate regions via initial sub-segmentation
Step 2: Use a greedy non-maximum suppression algorithm to recursively combine similar regions into larger ones
Step 3: Use the result from above to make final region proposals
Take the 2000 candidate regions and send them through a CNN (feature extractor).

Resize all regions to the same size (warped)

Convert to a feature vector (4096 dim) and feed to SVM for classification is the object present within the region?
Also, generate a bounding box.

Primary challenges with R-CNN:- Very slow as it has to classify 2000 regions per image



MASK-RCNN

Working-

Stage 1- feature extractor using resnet 101 this becomes input to RPN(works like a binary classifier object present or not) if region contains any object then that region becomes region of interest with multiple bbox containing objects

Stage 2- object detection-
Predicting class labels and bboc

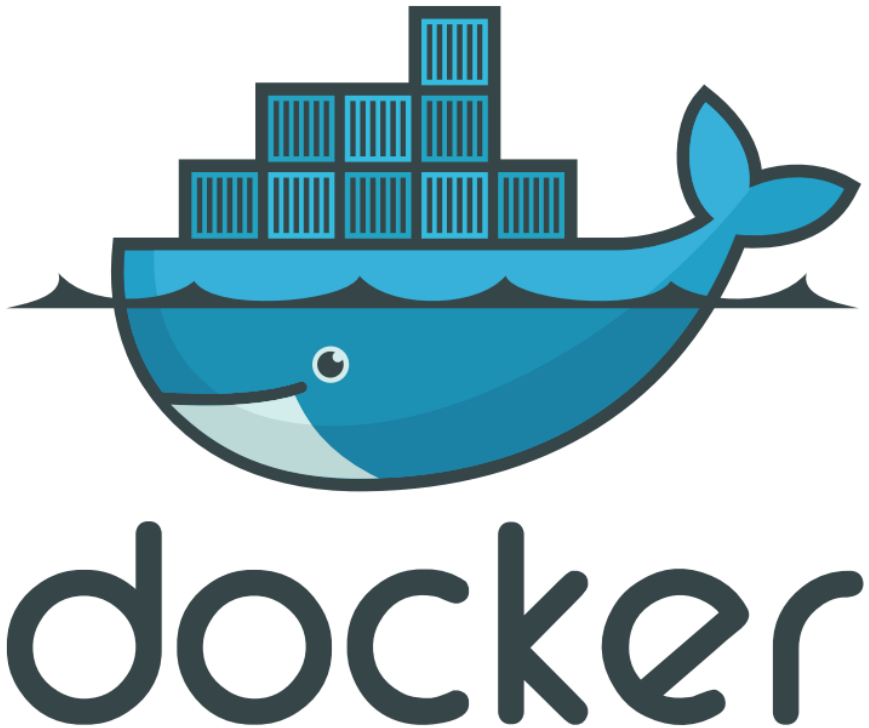
Stage 3 – Mask prediction-
Each pixel is given value 0/1 whether it is of the same class

Video Results-

Original

YOLO

Detectron2

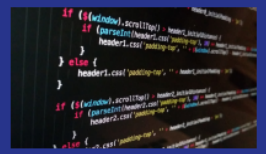

docker

What is a Docker?

- Docker is a containerization platform for developing, packaging, shipping, and running applications.

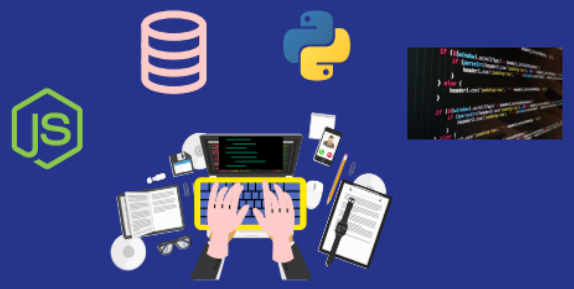
- It provides the ability to run an application in an isolated environment called a container.
- Makes deployment and development efficient.

Working Successfully



Let me test on my machine

Tester

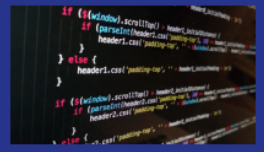


Developer

But it is working on my machine

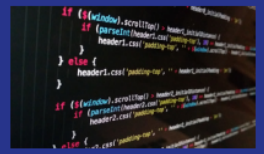
Tester

Tester



Developer

Tester



Developer

Developer

Dockerfile

Docker image

Docker Container

Docker Hub

Pull image

Docker image

Docker Container

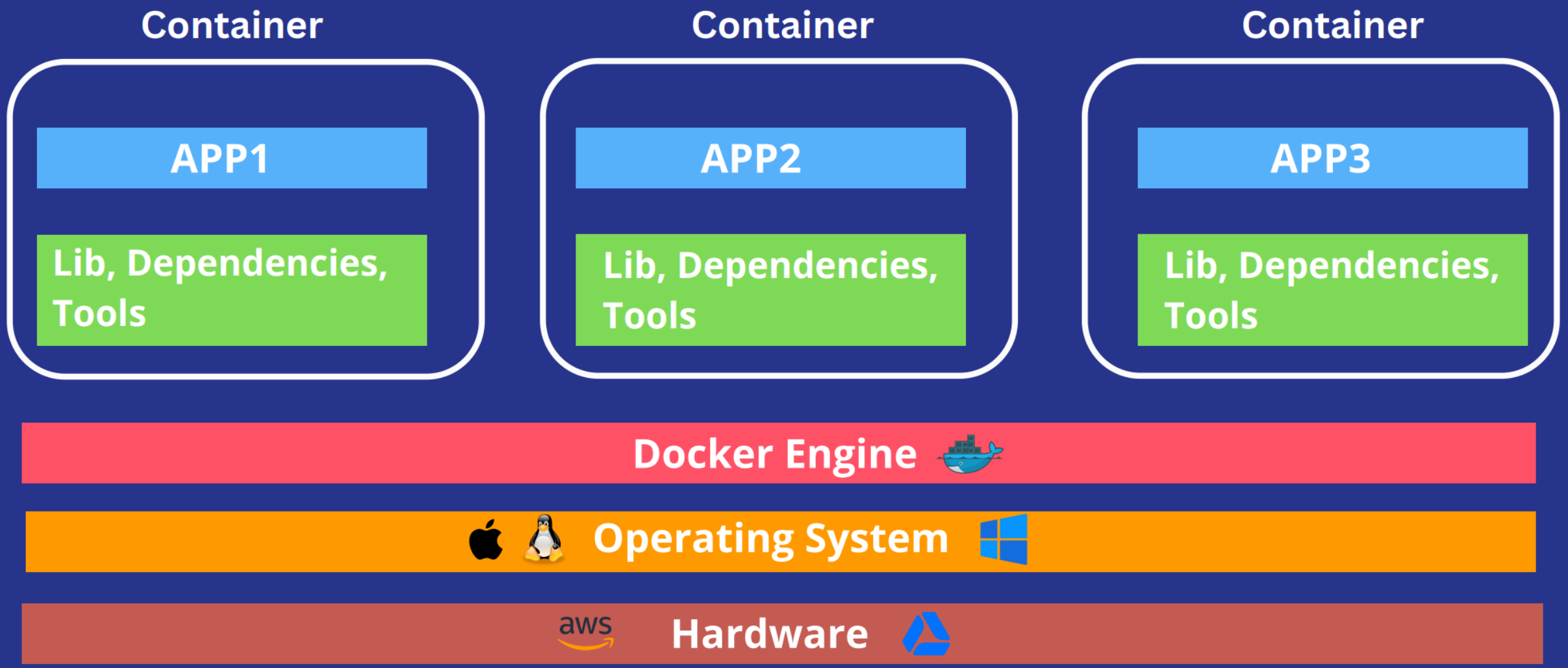
Staging Environment

Pull image

Docker image

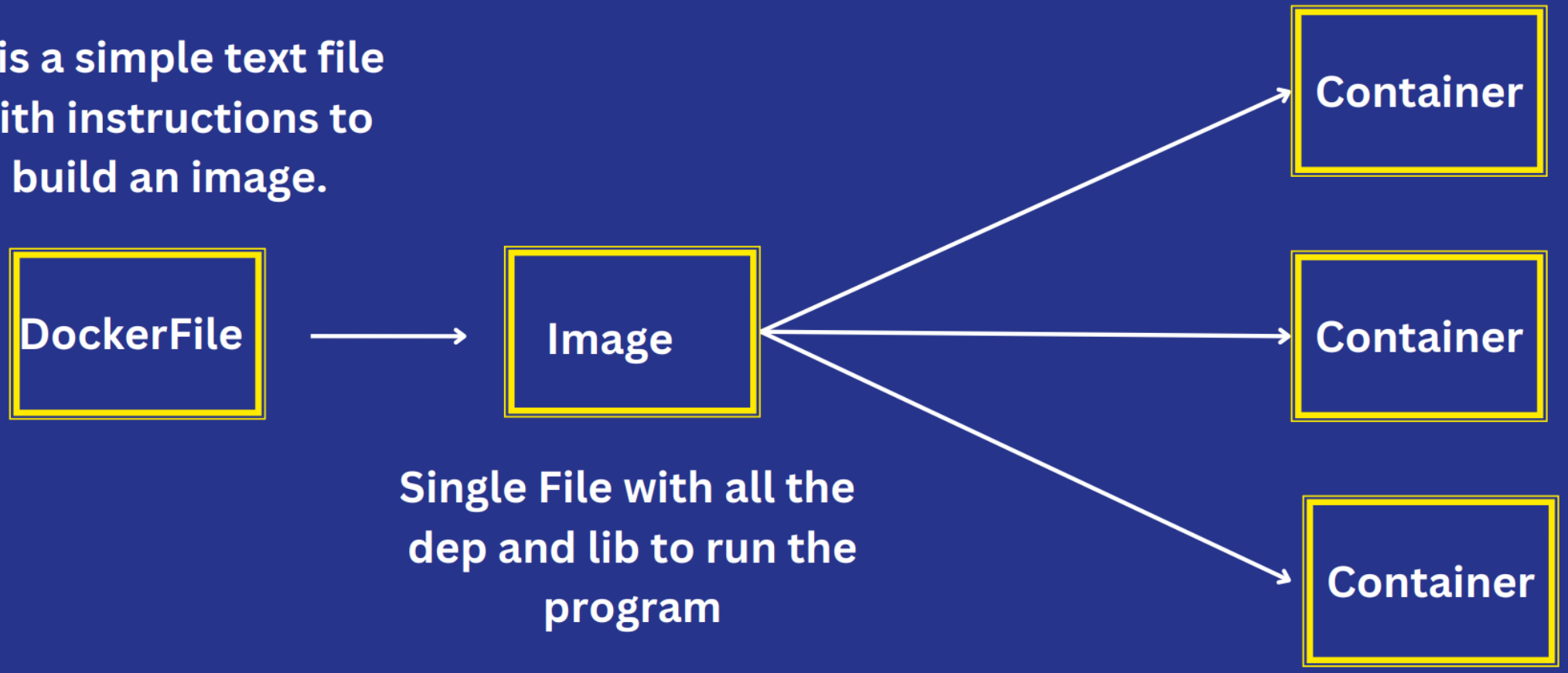
Docker Container

Test Environment


Container
APP1
Lib, Dependencies, Tools
Container
APP2
Lib, Dependencies, Tools
Container
APP3
Lib, Dependencies, Tools
Docker Engine
Operating System
aws
Hardware

Instance of an Image

It is a simple text file with instructions to build an image.



Single File with all the dep and lib to run the program

Creating a Docker Image

```
(.venv) C:\Users\Bhuvnesh's PC\PycharmProjects\pythonProject\computer vision\Cracks Dataset\docker streamlit>docker build -t bhuvneshsahu/crack-detection-app .
[+] Building 437.4s (8/9)                                                                                                  docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                                   0.0s
 => => transferring dockerfile: 284B                                                                                                   0.0s
 => [internal] load metadata for docker.io/library/python:latest                                                                       0.0s
 => [internal] load .dockerignore                                                                                                      0.0s
 => => transferring context: 2B                                                                                                        0.0s
 => CACHED [1/5] FROM docker.io/library/python:latest                                                                                  0.0s
 => [internal] load build context                                                                                                      0.1s
 => => transferring context: 3.53MB                                                                                                    0.1s
 => [2/5] COPY . /app                                                                                                                  0.4s
 => [3/5] WORKDIR /app                                                                                                                 0.0s
 => [4/5] RUN apt-get update && apt-get install -y libgl1-mesa-glx                                                                    23.7s
 => [5/5] RUN pip3 install -r requirements.txt                                                                                       413.1s
 => => #     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 823.6/823.6 kB 8.9 MB/s eta 0:00:00
 => => # Downloading nvidia_cudnn_cu12-8.9.2.26-py3-none-manylinux1_x86_64.whl (731.7 MB)
 => => #     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 731.7/731.7 MB 407.1 kB/s eta 0:00:00
 => => # Downloading nvidia_cufft_cu12-11.0.2.54-py3-none-manylinux1_x86_64.whl (121.6 MB)
```

Running the Image

```
(.venv) C:\Users\Bhuvnesh's PC\PycharmProjects\pythonProject\computer vision\Cracks Dataset\docker streamlit>docker run -p 8080:8080 bhuvneshsahu/crack-detection-app

Collecting usage statistics. To deactivate, set browser.gatherUsageStats to false.


  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8080
  Network URL: http://172.17.0.2:8080
  External URL: http://152.59.26.2:8080
```

Comparative Analysis Table

Metrices	SAM	Detectron2	YOLOv8
Precision	0.43	0.78	0.82
Recall	0.56	0.65	0.48
Total Loss	N/A	0.86	1.06
Box Loss	N/A	0.684	0.757
Segmentation loss	1.98	0.958	1.43
FPS	1	7	111
Inference time (MS)	230	40	9.5
No. of Trainable Parameters	91	44.1	3.2
Training time (Iter/Sec)	1.71(2)	4 (6)	3.5 (8)
Model size(MB)	357	334	6.5

- **Bounding Box Loss** calculates the error between the predicted and the ground truth boxes' geometry. It measures how well the model predicts the size and location of the bounding boxes.
- **Objectness Loss** determines how confident the model is about the presence of an object in the bounding box.
- **Segmentation Loss** quantifies how close the predicted segmentation map is to the ground truth map. It measures how effectively the model performs the semantic segmentation task.
- **Inference time:** how much time(ms) the model is taking to predict a single image of size 448*448.
- **Precision**: Out of all predicted pixels , how many are actually true
- **Recall:** Out of all True pixels , How many are predicted as True
- Total loss: Bbox loss + segmentation loss + objectness loss

Conclusion

- **Difficulty in custom training**

  YOLO < Detectron2 < SAM
- **For real time processing** - YOLO
- **For better segmentation** - Detectron2
- **For deployment purpose** - YOLO