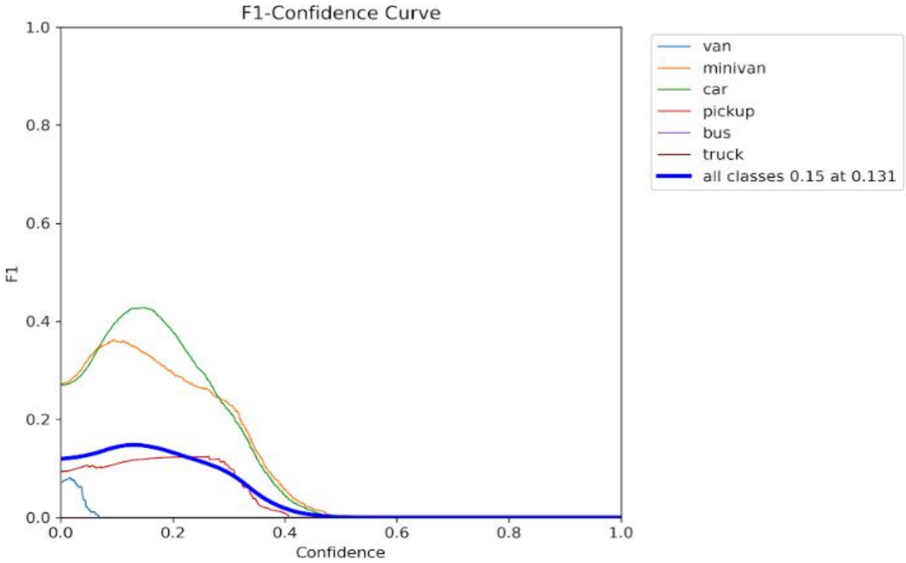
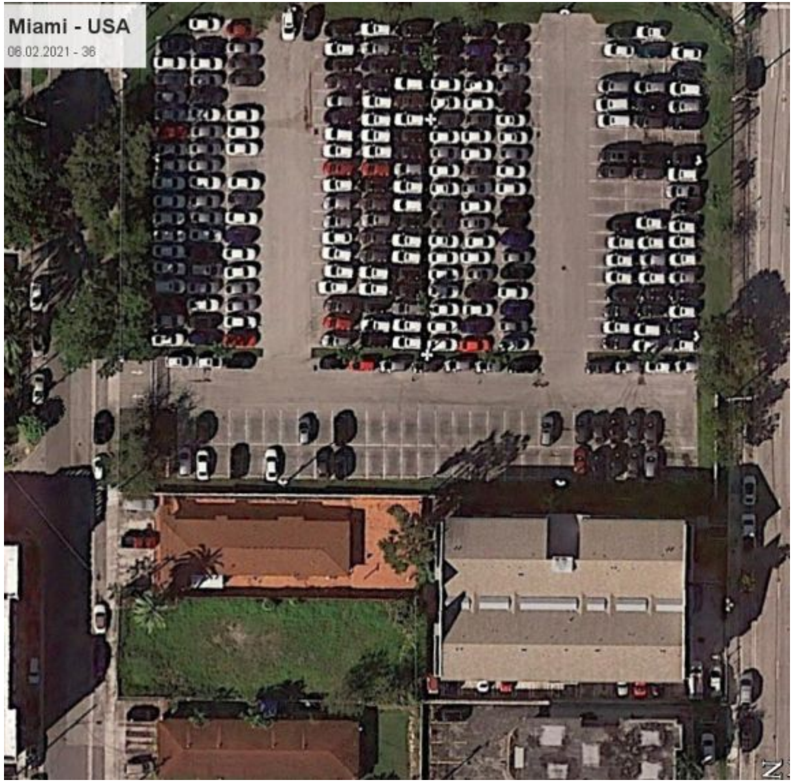
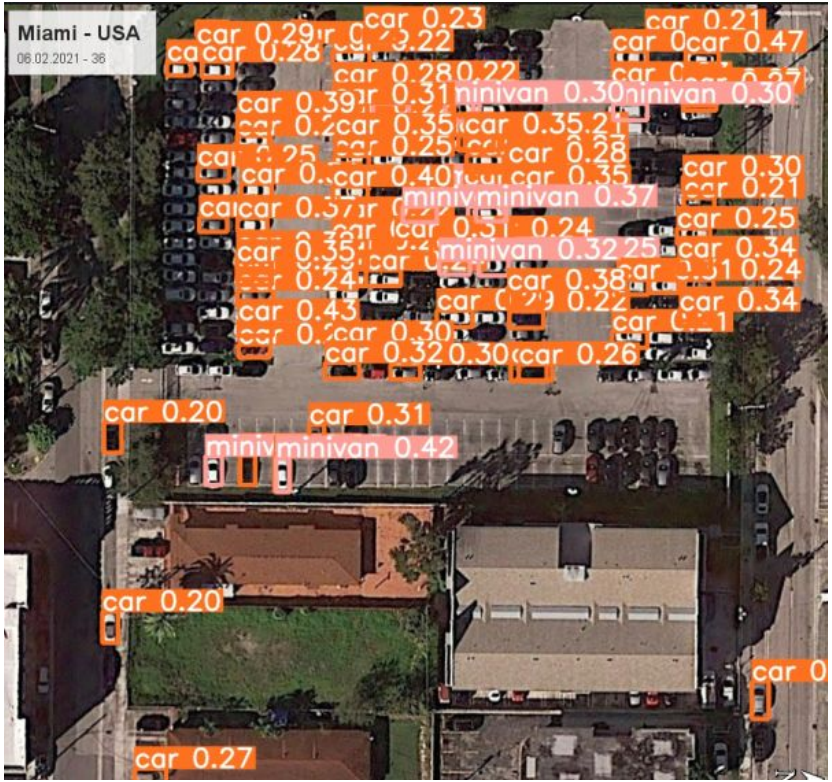
# Trial 1 (YOLOv8)

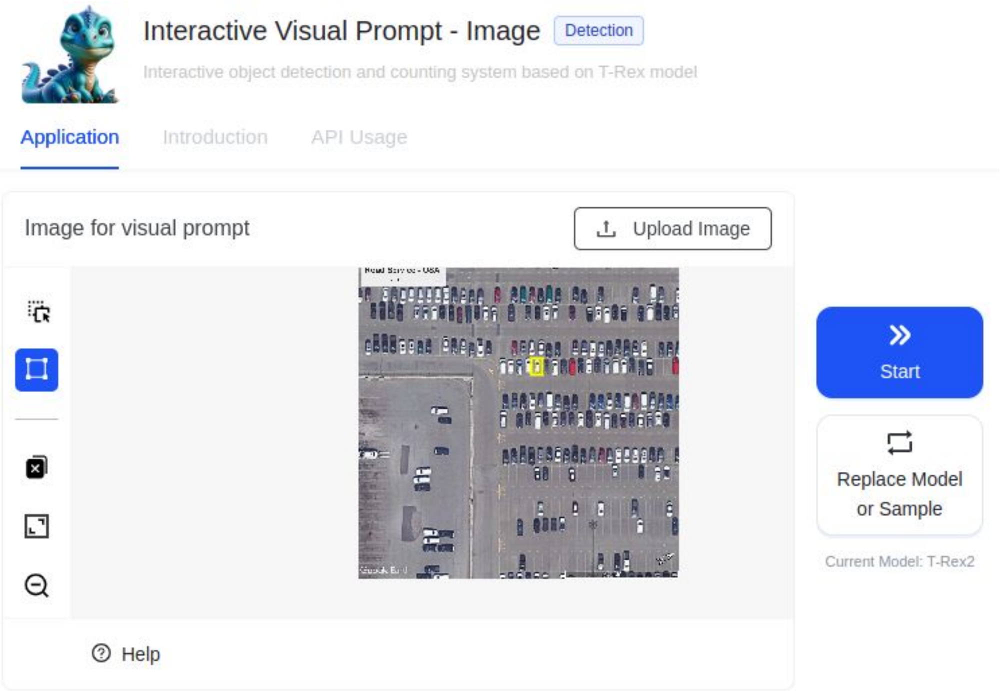



Miami - USA
06.02.2021 - 36

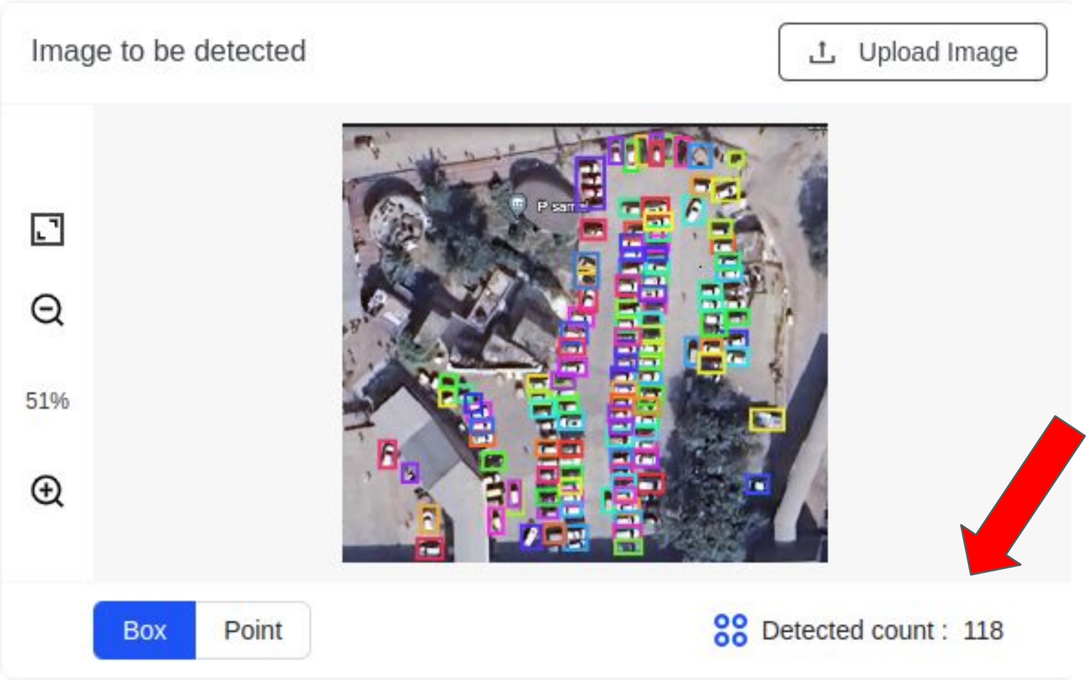

Miami - USA
06.02.2021 - 36
car 0.23
car 0.21
car 0.47
car 0.28
minivan 0.30
minivan 0.30
car 0.35
car 0.28
car 0.30
car 0.21
minivan 0.37
car 0.25
car 0.34
minivan 0.32
car 0.24
car 0.38
car 0.43
car 0.34
car 0.30
car 0.32
car 0.26
car 0.20
car 0.31
minivan 0.42
car 0.20
car 0.27

# Using Deep Data Space

https://deepdataspace.com/playground/ivp



# Result of “Interactive visual prompt”




Predicts 118 vehicles

# Using Interactive Visual Prompt (iVP) API

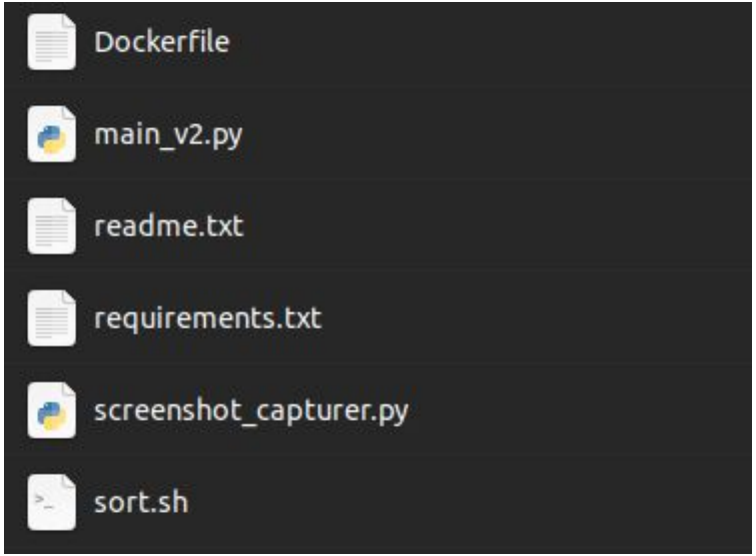
```python
# 1. Initialize the client with your API token.
from dds_cloudapi_sdk import Config
from dds_cloudapi_sdk import Client

token = "096ac96a78fea12d4e21a4372a89a944"
config = Config(token)
client = Client(config)

# 2. Upload local image to the server and get the URL.
#infer_image_url = "https://dev.deepdataspace.com/static/04_a.ae28c1d6.jpg"
infer_image_url = client.upload_file("/content/gdrive/My Drive/trex_api/107.jpg")  # you c
prompt_image_url = client.upload_file("/content/gdrive/My Drive/trex_api/test_5.png")  # u

# 3. Create a task with proper parameters.
from dds_cloudapi_sdk.tasks import IVPTask
from dds_cloudapi_sdk.tasks import RectPrompt
from dds_cloudapi_sdk.tasks import LabelTypes

task = IVPTask(
    prompt_image_url=prompt_image_url,
    prompts=[RectPrompt(rect=[0.826842, 0.337368, 0.020000, 0.047368], is_positive=True)],
    infer_image_url=infer_image_url,
    infer_label_types=[LabelTypes.BBox, LabelTypes.Mask],  # infer both bbox and mask
)

# 4. Run the task and get the result.
client.run_task(task)

# 5. Parse the result.
from dds_cloudapi_sdk.tasks.ivp import TaskResult

result: TaskResult = task.result

mask_url = result.mask_url  # the image url with all masks drawn on
objects = result.objects  # the list of detected objects

for idx, obj in enumerate(objects):
    # get the detection score
    print(obj.score)  # 0.42

    # get the detection bbox
    print(obj.bbox)  # [635.0, 458.0, 704.0, 508.0]

    # get the detection mask, it's of RLE format
    print(obj.mask.counts)  # ]o`f08fa14M3L2O2M2O1O1O1O1N2O1N2O1N2N3M2O3L3M3N2M2N3N1N2O...
```

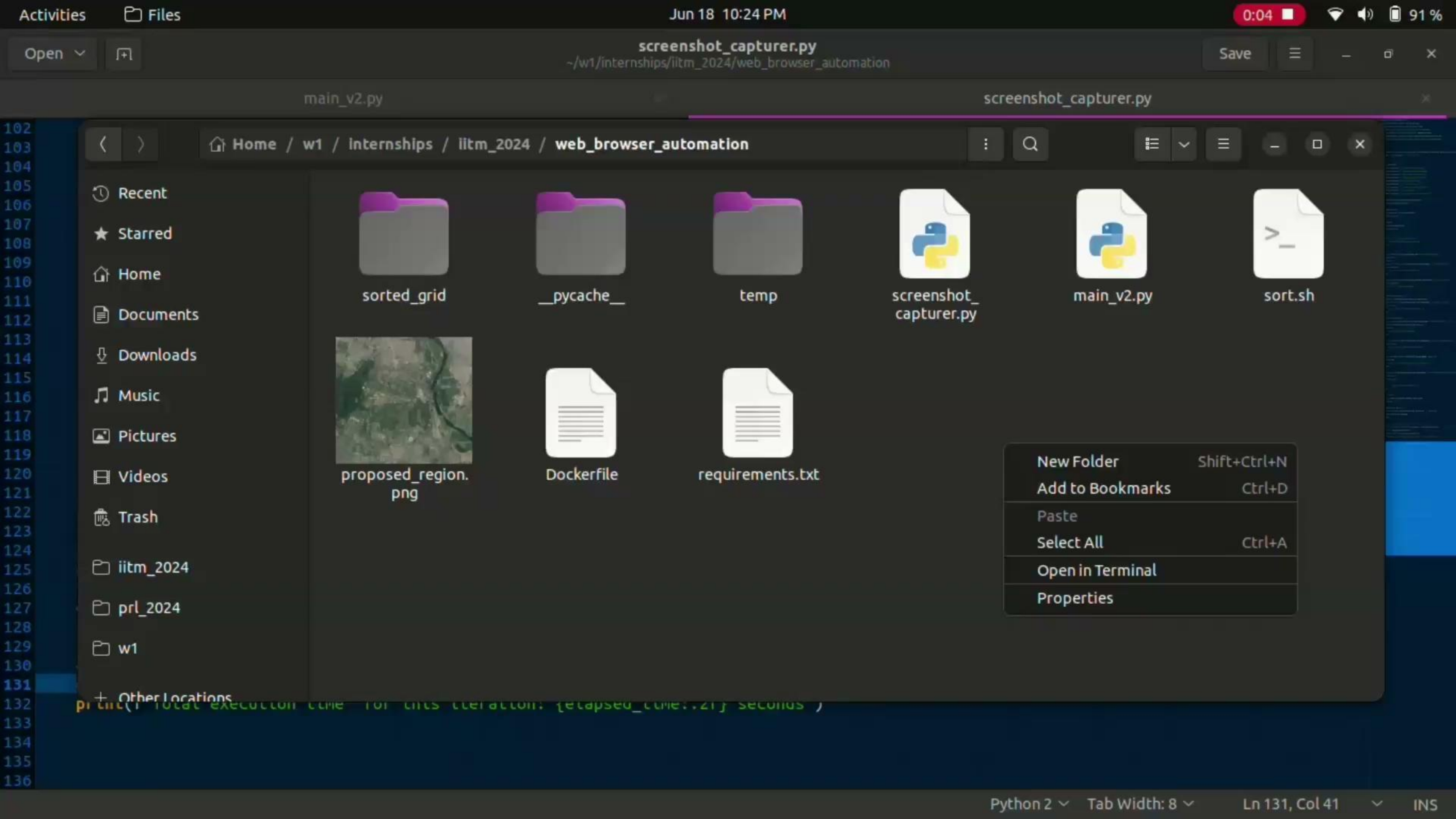
```
_XS79\m03M1O2O0O2O00000O1O00000000O01000
(950, 950)
0.34
[760.0, 559.0, 779.0, 607.0]

[18] len(objects)

239
```

Predicts 239 vehicles

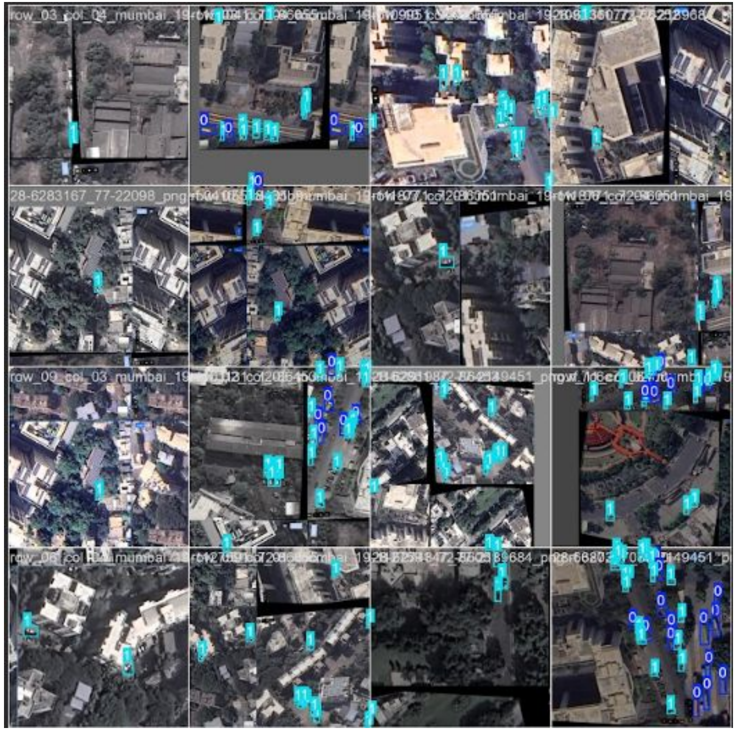

Sign in
Layers
Google
Imagery ©2024 Airbus, Maxar Technologies, Map data ©2024
India
3D Earth view is not available
Terms
Privacy
Send Product Feedback
10 m

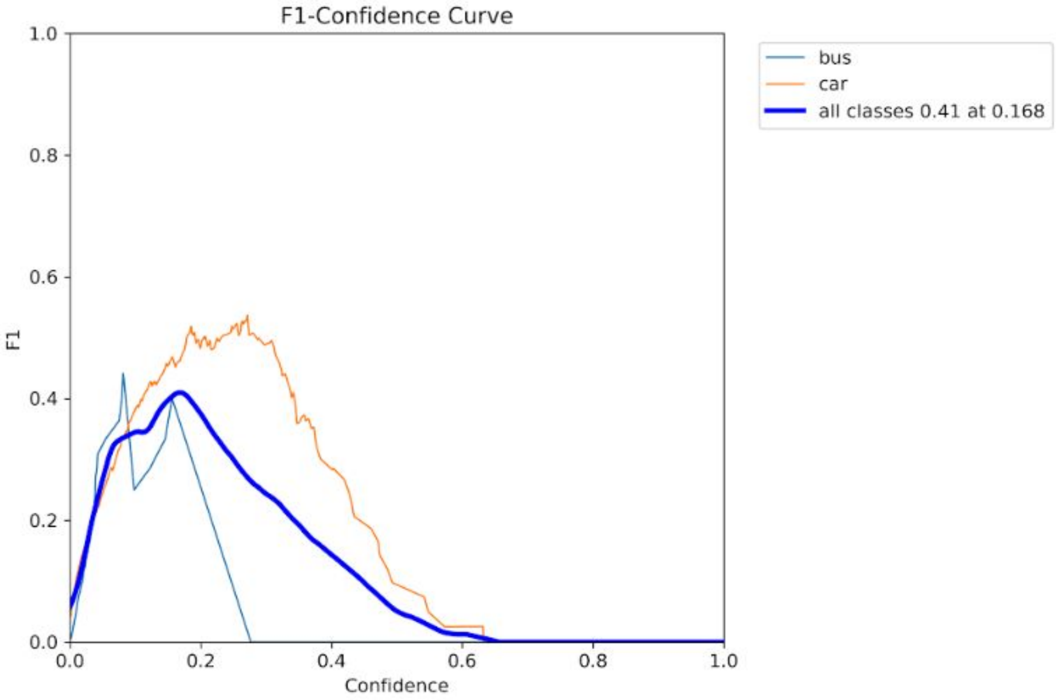
```python
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.action_chains import ActionChains
import time
import os
from tqdm import tqdm
from screenshot_capturer import capture_location_screenshots

# Inputs
del_lon = 0.0020116000  # change in longitude per snapshot
del_lat = 0.0008818999  # change in latitude per snapshot
central_lat, central_lon = 28.63272109, 77.21953181  # latitude & longitude for central C.P.
grid_dim = 1000  # in meters

def divide_grid(central_lat, central_lon, grid_dim, del_lon, del_lat):
    import math

    # Calculate the number of cells in each direction
    num_cells_lon = math.ceil(grid_dim / (del_lon * 111320))  # Converting degrees to meters
    num_cells_lat = math.ceil(grid_dim / (del_lat * 110540))  # Converting degrees to meters

    # Display necessary details
    print(f"Central latitude and longitude: {central_lat, central_lon}")
    print(f"Longitude change per cell: {del_lon:.8f} degrees")
    print(f"Latitude change per cell: {del_lat:.8f} degrees")
    print(f"Number of cells along zonal direction: {num_cells_lon} (i.e iterations)")
    print(f"Number of cells along meridional direction: {num_cells_lat} (i.e iterations)")
    print()

    # Calculate the final latitude and longitude of the top-left cell
    starting_lat = central_lat + ((num_cells_lat // 2) - 1) * del_lat
    starting_lon = central_lon - ((num_cells_lon // 2) - 1) * del_lon
```

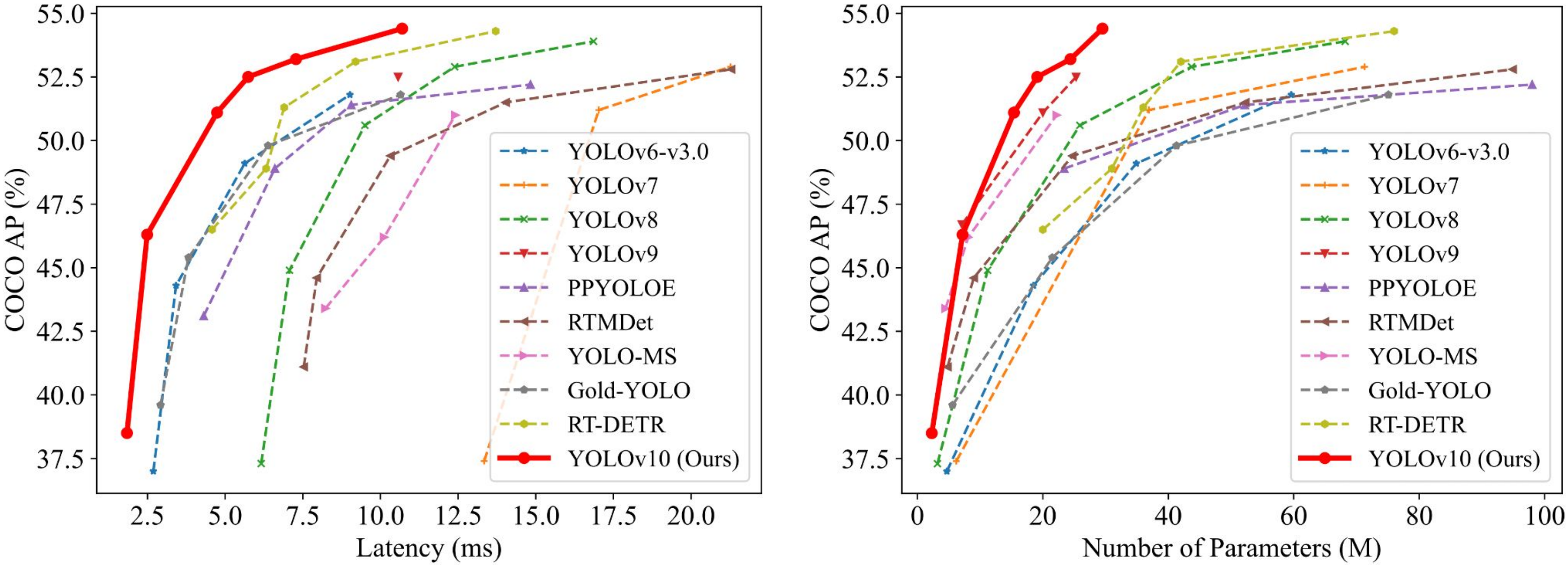



Activities
Files
Jun 18 10:24 PM
0:04
91 %
Open
screenshot_capturer.py
~/w1/internships/iitm_2024/web_browser_automation
Save
main_v2.py
screenshot_capturer.py
Home / w1 / internships / iitm_2024 / web_browser_automation
Recent
Starred
Home
Documents
Downloads
Music
Pictures
Videos
Trash
iitm_2024
prl_2024
w1
Other Locations
sorted_grid
__pycache__
temp
screenshot_capturer.py
main_v2.py
sort.sh
proposed_region.png
Dockerfile
requirements.txt
New Folder Shift+Ctrl+N
Add to Bookmarks Ctrl+D
Paste
Select All Ctrl+A
Open in Terminal
Properties
Python 2
Tab Width: 8
Ln 131, Col 41
INS

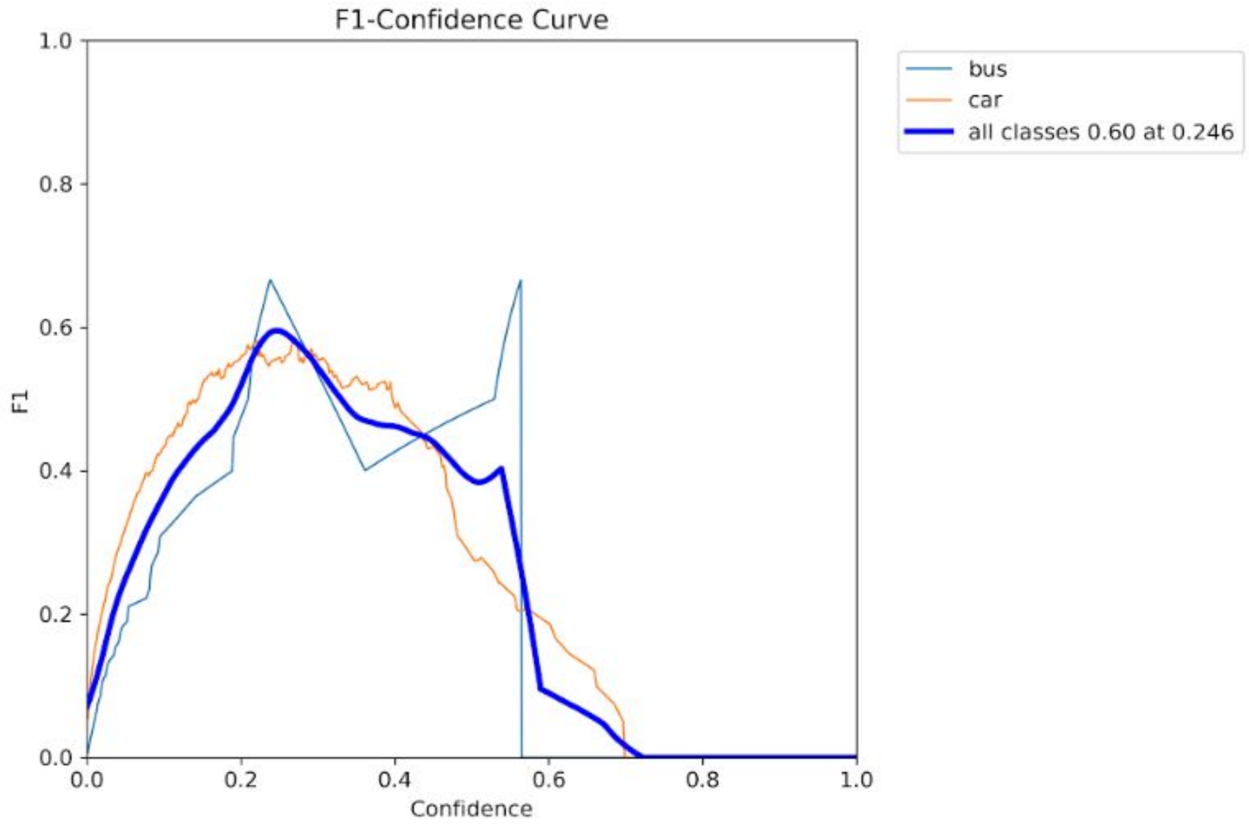
# Trial 2: YOLOv8





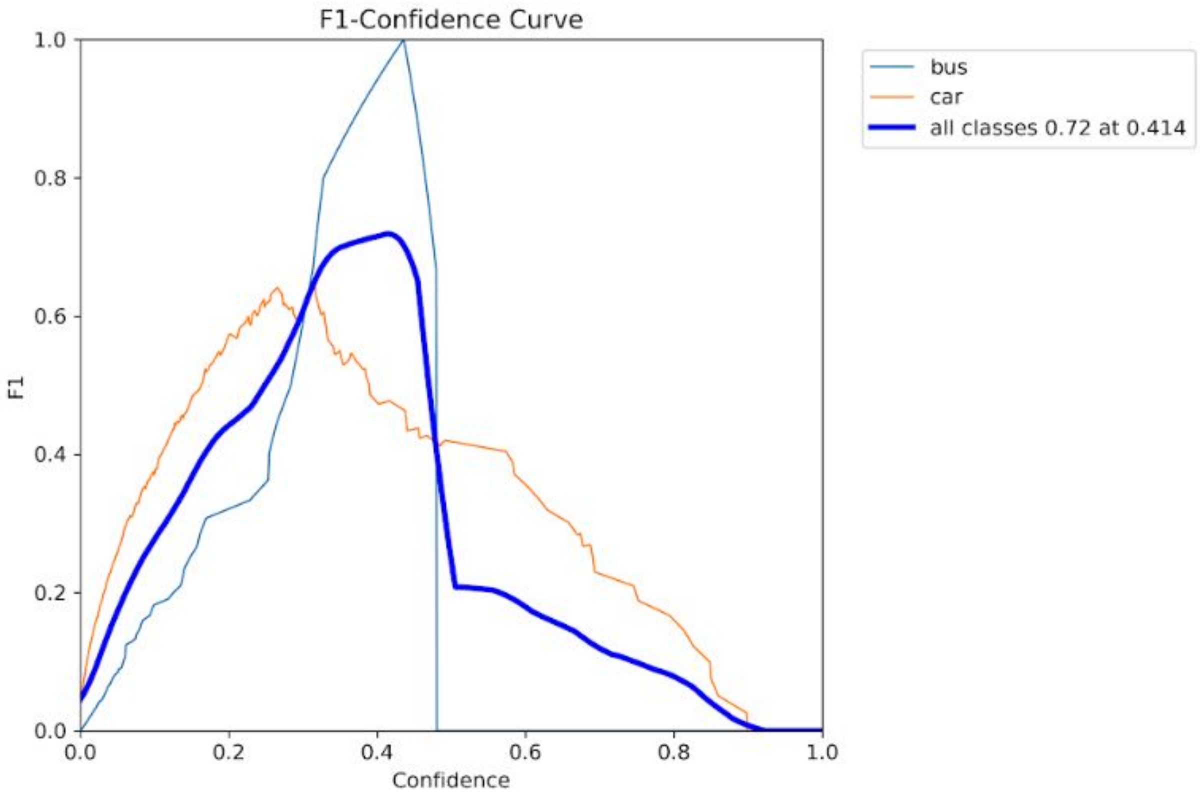

COCO AP (%)
Latency (ms)
Number of Parameters (M)
YOLOv6-v3.0
YOLOv7
YOLOv8
YOLOv9
PPYOLOE
RTMDet
YOLO-MS
Gold-YOLO
RT-DETR
YOLOv10 (Ours)

# YOLOv9



F1-Confidence Curve
1.0
0.8
0.6
0.4
0.2
0.0
F1
0.0
0.2
0.4
0.6
0.8
1.0
Confidence
bus
car
all classes 0.60 at 0.246

# YOLOv10

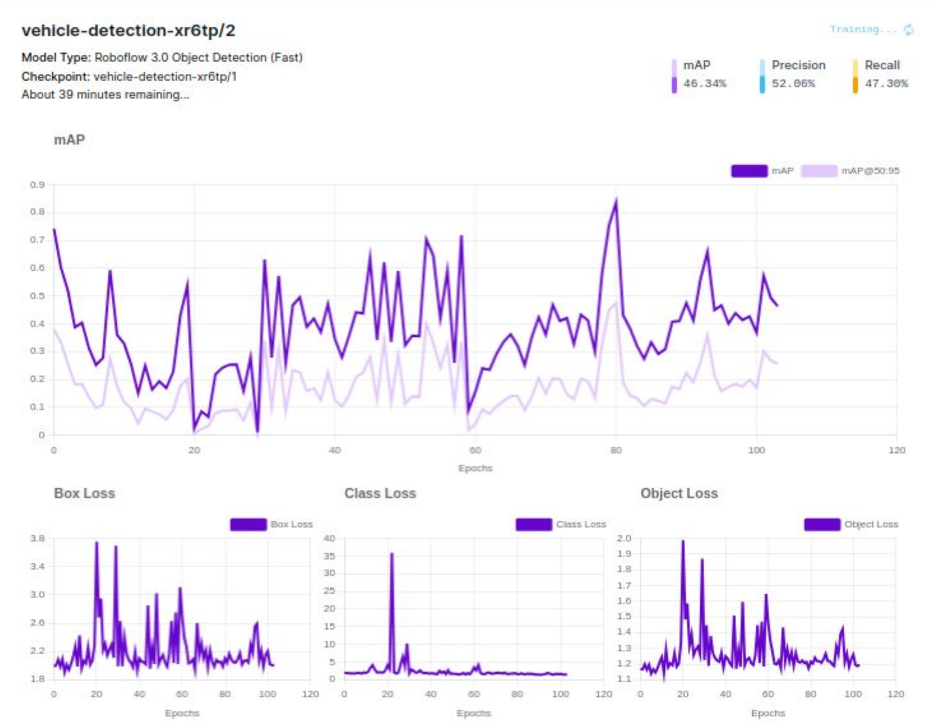

F1-Confidence Curve
1.0
0.8
0.6
0.4
0.2
0.0
F1
0.0
0.2
0.4
0.6
0.8
1.0
Confidence
bus
car
all classes 0.72 at 0.414

# Test runs


Sign in
car 0.46
car 0.35
car 0.89
car 0.81
bus 0.88
car 0.54
bus 0.61
car 0.90
car 0.78
bus 0.51
car 0.63
car 0.67
car 0.74
car 0.25
Google


bus 0.58
car 0.26
car 0.63
bus 0.96
car 0.35
car 0.44
car 0.53
car 0.74
car 0.21
car 0.23
car 0.67
car 0.80
car 0.72
bus 0.96
car 0.26
car 0.59
car 0.85
car 0.81
car 0.65
car 0.87
car 0.89
car 0.60
car 0.56
car 0.24
car 0.50
car 0.61
car 0.76
car 0.66
Google

# Roboflow (R-CNN and YOLOv)





# Conclusion:

- Overall Best Model: YOLOv8 seems to have a higher number of true positives for both car and background classes. Despite having more false negatives for buses and backgrounds, it balances with a significantly higher true positive rate for cars, which might be more critical depending on the application.

- For Specific Use Cases:

  - If identifying cars correctly is crucial, YOLOv8 is the best choice.

  - If minimizing false negatives for buses is critical, YOLOv9 is preferable.

  - **If a balanced performance for cars and background with lower false negatives is essential, YOLOv10 may be considered.**

Depending on the specific priorities and requirements of the application, you can choose the most suitable model based on this analysis.

```python
with Dataset(output_nc_file, 'w', format='NETCDF4') as ncfile:

    ncfile.createDimension('records', len(data))


    lats = ncfile.createVariable('lat', 'f4', ('records',))
    lons = ncfile.createVariable('lon', 'f4', ('records',))
    cars = ncfile.createVariable('cars', 'i4', ('records',))
    buses = ncfile.createVariable('buses', 'i4', ('records',))


    for i, (lat, lon, num_cars, num_buses) in enumerate(data):
        lats[i] = lat
        lons[i] = lon
        cars[i] = num_cars
        buses[i] = num_buses

print(f"Results saved to {output_nc_file}")
```

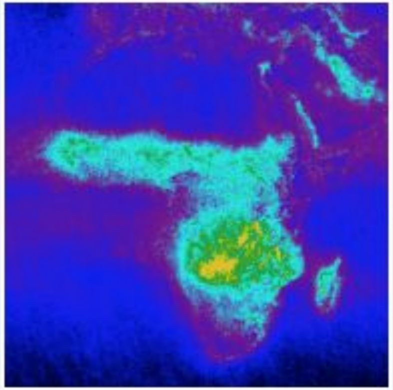
```
image 1/2 /content/gdrive/My Drive/iitm_2024/yolov10/test_runs/19.1118771_72.86655_129m.png: 608x640 11 buss, 91 cars, 17.2ms
image 2/2 /content/gdrive/My Drive/iitm_2024/yolov10/test_runs/19.1154041_72.8605151_129m.png: 608x640 6 buss, 28 cars, 9.9ms
Speed: 3.2ms preprocess, 13.5ms inference, 0.6ms postprocess per image at shape (1, 3, 608, 640)
Results saved to /content/gdrive/My Drive/iitm_2024/yolov10/test_runs/detection_results.nc
```

```
shubham@shubham-IdeaPad-3-14IIL05:~/w1/work/iitm_2024$ ncdump detection_results.nc
netcdf detection_results {
dimensions:
        records = 2 ;
variables:
        float lat(records) ;
        float lon(records) ;
        int cars(records) ;
        int buses(records) ;
data:

 lat = 19.11188, 19.1154 ;

 lon = 72.86655, 72.86051 ;

 cars = 91, 28 ;

 buses = 11, 6 ;
}
shubham@shubham-IdeaPad-3-14IIL05:~/w1/work/iitm_2024$
```

# Study region:

10 km X 10 km

# Sentinel-5P OFFL HCHO: Offline Formaldehyde



**Dataset Availability**

2018-12-05T12:14:36Z–2024-07-07T16:21:59Z

**Dataset Provider**

European Union/ESA/Copernicus

**Earth Engine Snippet**

```
ee.ImageCollection("COPERNICUS/S5P/OFFL/L3_HCHO")
```

**Tags**

air-quality bira copernicus dlr esa eu formaldehyde hcho pollution s5p sentinel tropomi

Description Bands Image Properties Terms of Use

**Resolution**

1113.2 meters

# Bands:

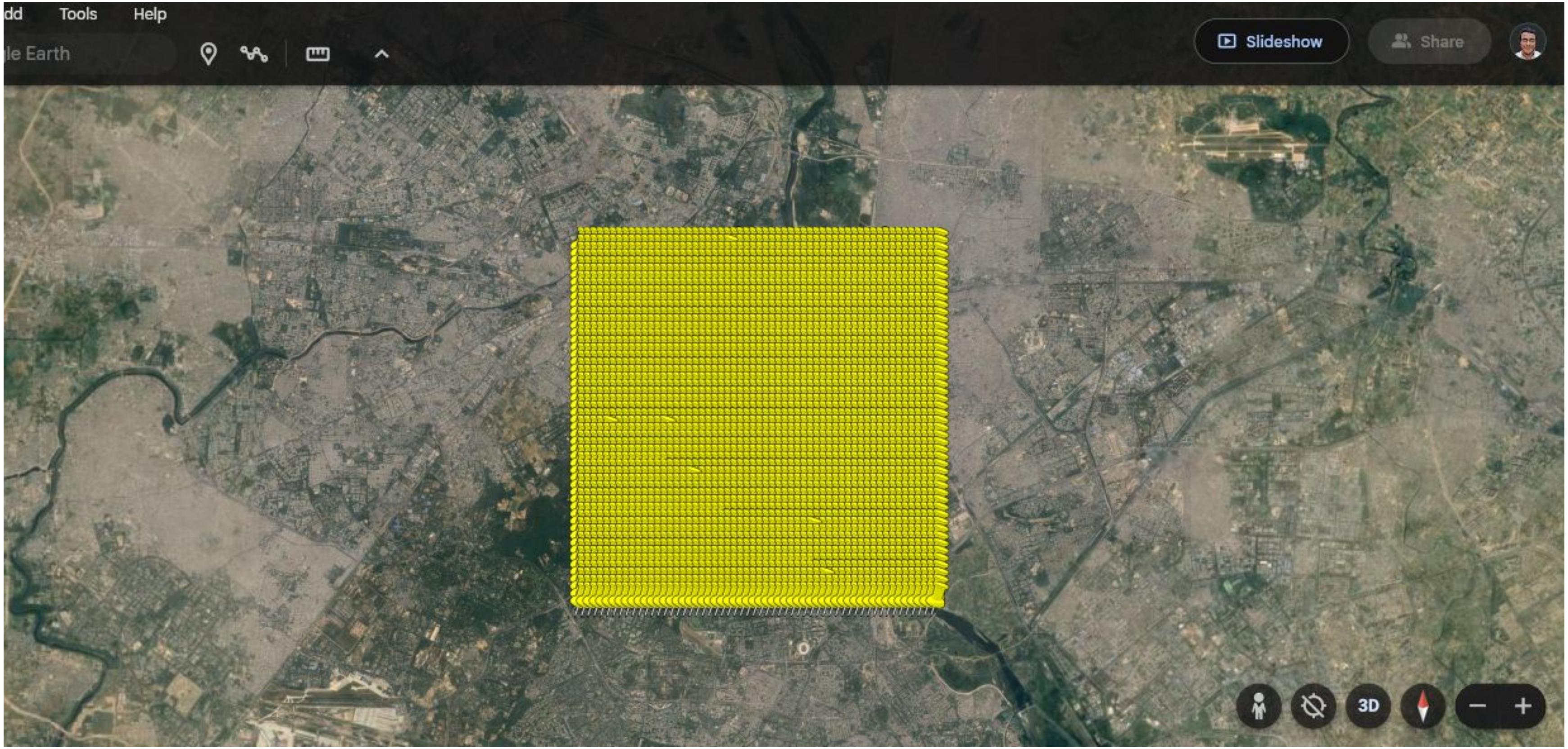
| Name | Units | Min | Max |
|---|---|---|---|
| tropospheric_HCHO_column_number_density | mol/m^2 | -0.0172* | 0.0074* |
| tropospheric_HCHO_column_number_density_amf | mol/m^2 | 0.177* | 4.058* |
| HCHO_slant_column_number_density | mol/m^2 | -0.01425* | 0.00735* |
| cloud_fraction | Fraction | 0* | 1* |
| sensor_azimuth_angle | deg | -180* | 180* |
| sensor_zenith_angle | deg | 0.098* | 66.57* |
| solar_azimuth_angle | deg | -180* | 180* |
| solar_zenith_angle | deg | 8.76* | 101.17* |

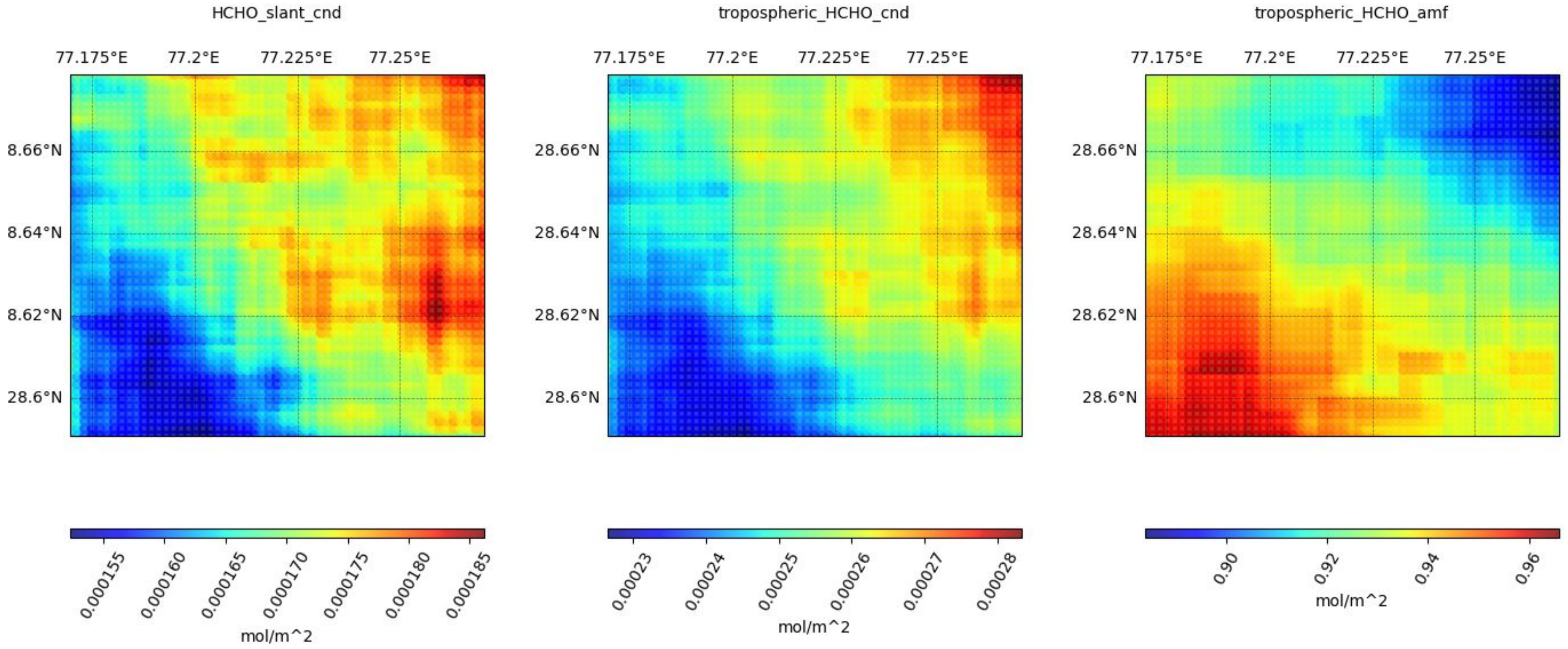
draft_1 *

Get Link | Save | Run | Reset | Apps

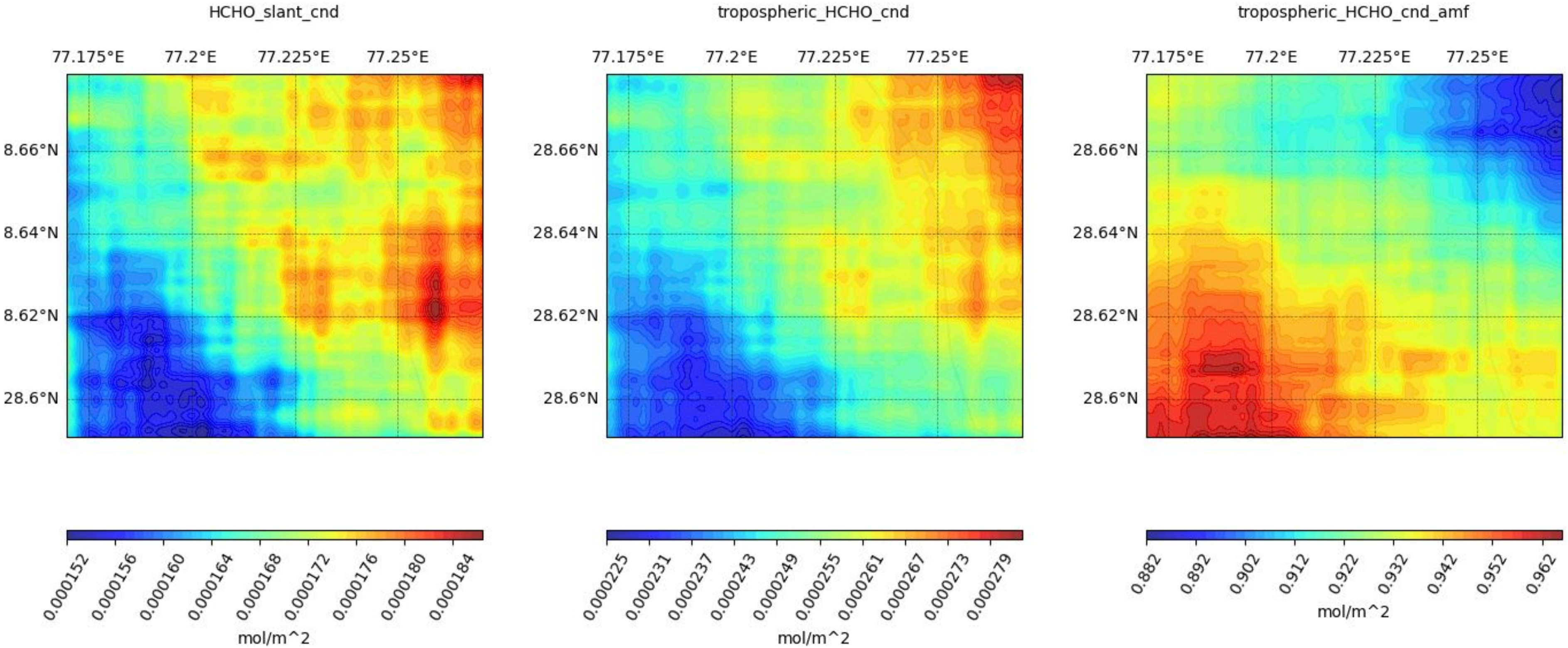
```
var center = ee.Geometry.Point([77.2200000, 28.6338889]);

var roi = center.buffer(5000).bounds();

var startDate = '2021-01-01';
var endDate = '2021-12-31';
var dateRange = ee.DateRange(startDate, endDate);

// Load the Sentinel-5P HCHO dataset and filter by date range
var s5p_HCHO = ee.ImageCollection('COPERNICUS/S5P/NRTI/L3_HCHO')
                .filterDate(dateRange)
                .select(['tropospheric_HCHO_column_number_density',
                         'tropospheric_HCHO_column_number_density_amf',
                         'HCHO_slant_column_number_density']);


var annualMeanHCHO = s5p_HCHO.mean();

var sampled = annualMeanHCHO.sample({
  region: roi,
  scale: 200,
  geometries: true
});

print('Annual mean HCHO values for each pixel in the ROI:', sampled);

Export.table.toDrive({
  collection: sampled,
  description: 'Annual_Mean_HCHO_Values_Per_Pixel_1km_Resolution',
  fileFormat: 'CSV'
});
```

# .kml file




HCHO_slant_cnd
tropospheric_HCHO_cnd
tropospheric_HCHO_amf
77.175°E
77.2°E
77.225°E
77.25°E
8.66°N
8.64°N
8.62°N
28.6°N
28.66°N
28.64°N
28.62°N
0.000155
0.000160
0.000165
0.000170
0.000175
0.000180
0.000185
mol/m^2
0.00023
0.00024
0.00025
0.00026
0.00027
0.00028
mol/m^2
0.90
0.92
0.94
0.96
mol/m^2

# Cubic interpolation



# Bilinear interpolation