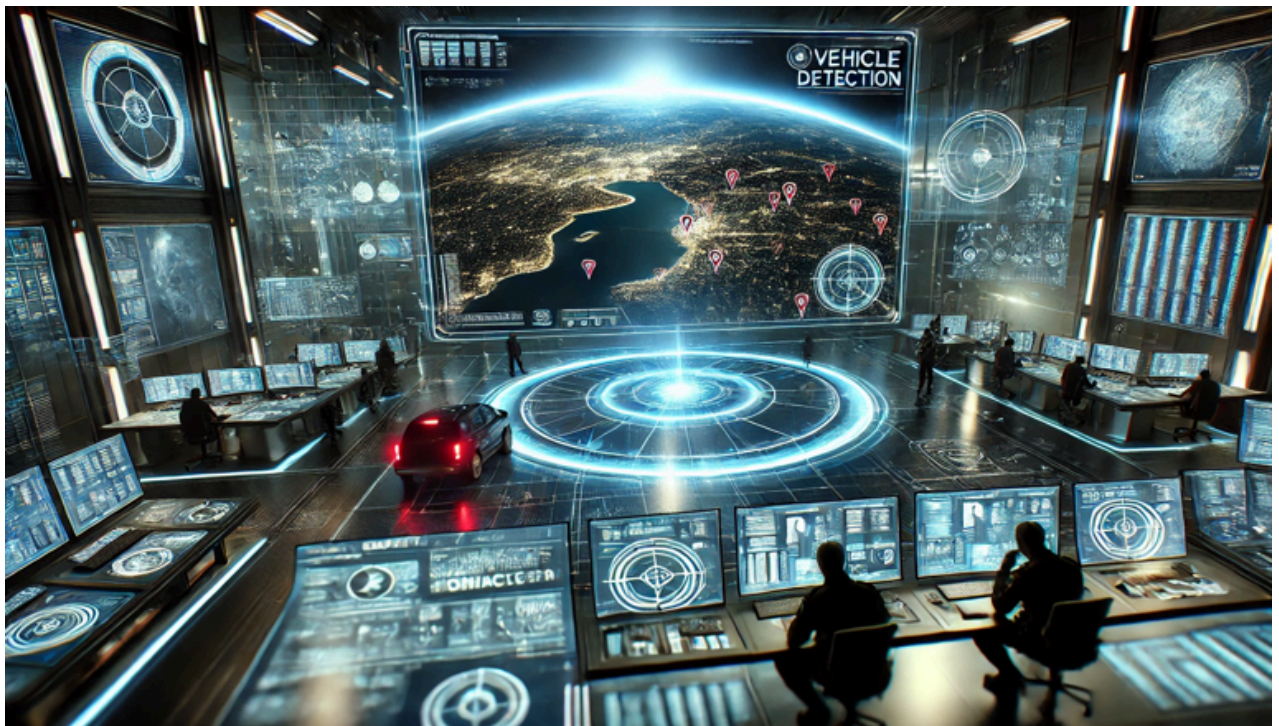


Introduction

After conquering the culinary challenge and designing a menu worthy of Michelin stars, you're more determined than ever to apply for the researcher role at the *real* ICCS. But as fate would have it, your fingers once again betrayed you on the application form. Instead of submitting to the Institute of Communication and Computer Systems, you've accidentally applied to the International Computing Cluster of Spies!

Now, you find yourself in an undisclosed, high-security bunker surrounded by agents in dark suits and tinted glasses. The ICCS team briefs you on their latest operation: they've detected unusual vehicle movements in several key regions around the globe. Suspicious convoys, secret rendezvous, and unexplained patterns are raising alarms, and they need immediate intel to understand what's happening. Fortunately, they've just acquired *satellite images* from the *xView* mission, capturing snapshots of these areas.

With time running out, the pressure is on. The satellite is scheduled to pass over these regions again soon, and they need a sophisticated methodology to quickly and accurately count the number of vehicles present. This intel is crucial — it could determine the success of a high-stakes international operation. The ICCS team is relying on your expertise to not only detect the vehicles but also to provide insights (vehicle counts) that could turn the tide of this covert mission. No pressure!



Task

Given the satellite images from previous xView missions over various regions, and the associated annotations provided in COCO format, your objective is to design a *deep learning-based* methodology capable of accurately detecting and counting vehicles in future satellite images to be captured by the xView mission. The system should differentiate between *small* and *large vehicles* and provide reliable counts for both categories.

Dataset Description

The xView dataset is a large satellite image dataset, specifically designed for object detection tasks. It features high-resolution images with a spatial resolution of 0.3 meters per pixel. Each image is $\sim 2500 \times 3000$ pixels in size and provided as an 8-bit, 3-channel RGB image. In this version, the images have been pre-processed to generate smaller, overlapping patches of 640×640 pixels, covering the following 23 classes:

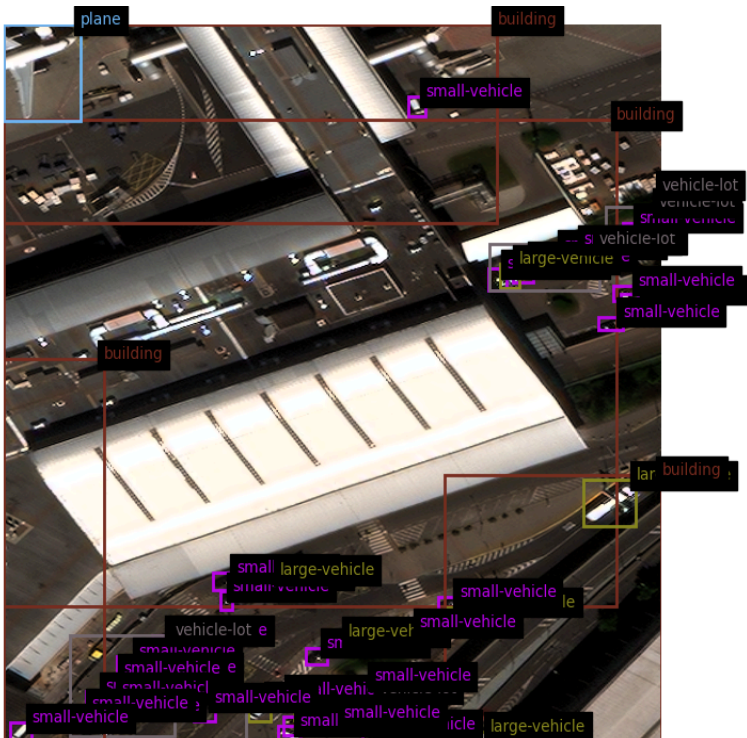
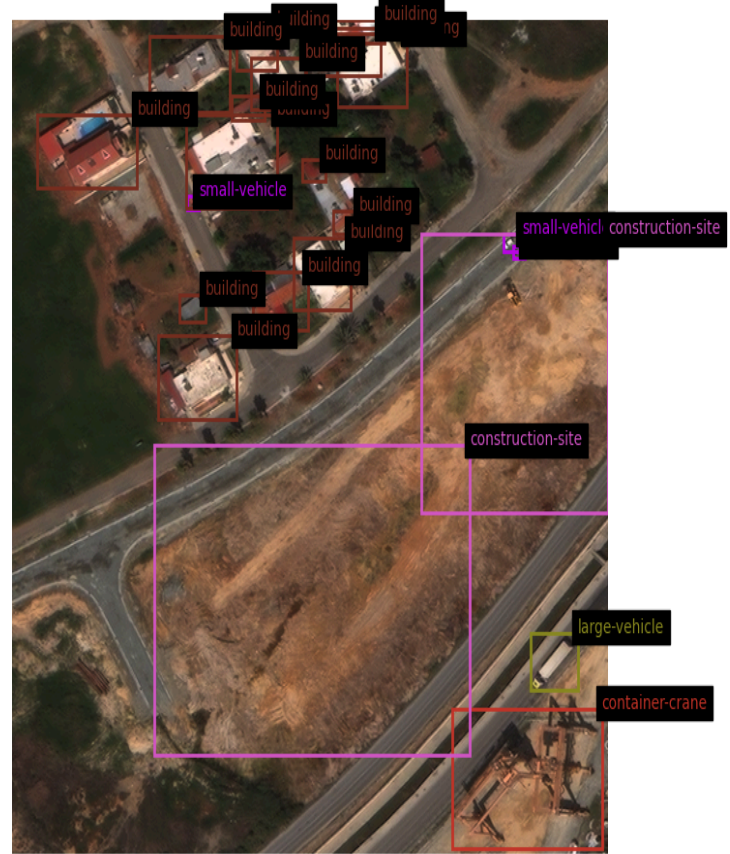
ID	Name	ID	Name
0	building	12	helicopter
1	small-vehicle	13	aircraft-hanger
2	large-vehicle	14	shed
3	plane	15	damaged-building
4	locomotive	16	pylon
5	vehicle-lot	17	ship
6	storage-tank	18	container
7	shipping-container-lot	19	hut/tent
8	tower	20	container-crane
9	trailer	21	construction-site
10	facility	22	railway-vehicle
11	helipad		

Structure

The dataset consists of a folder named *images*, containing all the satellite images in *tif* format. Each image is named sequentially, e.g., xView_001_1.tif, xView_001_2.tif, xView_002_1, etc., following the format xView_{image_index}_{patch_index}. Alongside this folder, there is a separate JSON file (*annotations.json*) that holds the annotations in COCO format, including details about the images, bounding boxes, and object categories.

Additionally, there is a second folder named *future_pass_images*, which contains satellite images simulating the upcoming pass of the xView mission over the regions of interest. These images do not have annotations, as they represent the data to be processed for the current task.

NOTE: For the image and patch indices, zero-indexing has been employed.



JSON Annotations File Description

1. Info Section:

Provides metadata about the dataset.

- **description:** A short description of the dataset (e.g., "Xview subimages").
- **version:** The version of the dataset (e.g., "1.0").
- **year:** The year the dataset was created or released (e.g., 2024).
- **contributor:** The entity who contributed to the dataset (e.g., "Mainland").
- **date_created:** The date the dataset was created (e.g., "2024/10/03").

2. Licenses Section:

Contains license information about the dataset.

- **url:** The URL to the license (set to null in this dataset).
- **id:** The ID of the license (set to null).
- **name:** The name of the license (set to null).

3. Images Section:

A list of image metadata entries.

- **id:** A unique identifier for the image.
- **file_name:** The filename of the image (e.g., "xView_1694_7.tif").
- **height:** The height of the image in pixels (e.g., 640).
- **width:** The width of the image in pixels (e.g., 640).
- **date_captured:** The date the image was captured.
- **dataset:** The name of the dataset the image belongs to (e.g., "xView").

4. Annotations Section:

Contains annotations for objects within the images, including bounding boxes and segmentation information.

- **id:** A unique identifier for the annotation.
- **image_id:** Links the annotation to the image using the id from the images section.
- **category_id:** A numeric identifier linking the object to a category from the categories section.
- **bbox:** The bounding box for the object in the format [x, y, width, height], where x and y are coordinates, and width and height represent the size of the bounding box.
- **area:** The area of the bounding box.
- **segmentation:** An array describing the segmentation of the object (if existing).
- **iscrowd:** A binary flag indicating whether the object is part of a crowd (0 means it is not a crowd).

5. Categories Section

Contains information about the object categories in the dataset.

- **id:** A unique identifier for the category.
- **name:** The name of the category (e.g., "building," "vehicle").
- **supercategory:** A broader category grouping related categories







together





Performance Evaluation


The performance of your implemented methodology will be evaluated based on how accurately it counts the number of **small** and **large vehicles** in each image from the *future_pass_images* folder. For each image, the goal is to correctly count all instances of both **small** and **large vehicles**.

The closer your model's vehicle counts are to the actual number of vehicles in the images, the better the performance will be considered.

Implementation Guidelines

1.  **Write and Execute Your Code in Jupyter Notebook:**
 - All code should be implemented and executed within a Jupyter notebook. Ensure your notebook is well-organized and easy to follow.
2.  **Choose Your Deep Learning Model:**
 - You are free to select any deep learning model that you find suitable for the task. Whether it's a pre-trained model or a custom architecture, the choice is yours.
3.  **Framework Flexibility:**
 - You may use either PyTorch or TensorFlow/Keras for model implementation and training. Choose the framework you're most comfortable with or find the most effective for this task.
4.  **Comment Your Code:**
 - Clearly explain all functions and important parts of your code with comments. This will help reviewers understand your thought process and decisions throughout the implementation.
5.  **Data Preprocessing and Splitting:**
 - You are free to preprocess the given dataset in any way you see fit. This includes resizing, augmenting, or normalizing the images to prepare them for model training.
 - Feel free to split the dataset into any relevant sub-datasets, to optimize your model's performance.
6.  **Metrics and Evaluation:**
 - While implementing your model, consider the performance metrics provided (e.g. IoU, mAP, precision, recall, etc).

7.  **Optimization and Hyperparameters:**
 - You are free to experiment with different optimization techniques, such as learning rate schedules, data augmentations, and regularization methods.
8.  **Clearly Define Assumptions:**
 - If you make any assumptions regarding the dataset or task, please clearly define them in the notebook.
9.  **Use External Libraries:**
 - You are free to use any external libraries that are relevant to the task, such as libraries for data augmentation, visualization, or deep learning.
10.  **Visualizations (Optional):**
 - If probable, provide visualizations of your results, such as detection overlays on the images, training curves, etc., to enhance clarity.

Good luck! 

Submission

The final submission must be **one (1) .zip file** including the following:

1. The **executed Jupyter Notebook** with all the outputs in the cell,
2. The **model's weights** either in *.pt* for PyTorch models or *.zip* for Tensorflow models,
3. The **output JSON file** (explained below).

To submit your results, please prepare a JSON file following the structure provided below. Your output should contain:

- An array called ***results*** with an entry for each image in the *future_pass_images* folder. For each image, include the *image_id* (the filename of the image) and provide the counts for both *small_vehicle_count* and *large_vehicle_count*, representing the number of small and large vehicles detected in that image.
- In addition, include an *overall_metrics* section where you report the average performance metrics (e.g. IoU, mAP, precision, recall, etc.) of your model during the training phase.

Make sure the JSON file adheres to the following structure and that your reported counts and metrics are accurate.

Example JSON File

```
{
  "results": [
    {
      "image_id": "xView_001_0.tif", // Filename of the image
      "small_vehicle_count": 15, // Total count of small vehicles detected
      "large_vehicle_count": 5 // Total count of large vehicles detected
    },
    {
      "image_id": "xView_002_2.tif",
      "small_vehicle_count": 12,
      "large_vehicle_count": 8
    }
  ]
  // Repeat for each image in future_pass_images folder
},
"overall_metrics": {
  "train": {
    "average IoU": 0.86, // Average IoU across all train images
    "average_mAP": 0.77, // Average mAP across all train images
    "average_precision": 0.78, // Average bounding box precision
    // across train images
    "average_recall": 0.83 , // Average bounding box recall across
    // train images
    "metric_5": 0.23 // Numerical value of metric_5 across
    // train images
  }
}
```

NOTE: This is an example template of the results json file. Your final json file may include more or less entries in the overall_metrics!