AAS 23-306

SPACE INSPECTION FOR CUBESATS AND OTHER SPACE TARGETS USING OBJECT DETECTION IN IMAGES

Anh Quoc Nguyen* **and Donghoon Kim†**

As the number of space debris and satellites increases, they are beginning to pose threats to space missions. Therefore, it is becoming more important to increase the situational awareness capability of satellites. Thus, it is essential to have an accurate, fast, and lightweight computer vision model deployed on spacecraft for space inspection. However, the biggest challenge in training such a supervised model is the lack of real data, and this study proposes a solution to this problem. As a first step, the authors designed and implemented a rendering pipeline to generate, customize, and randomize images of space targets with space backgrounds. A study was conducted to find the configurations of the generator that produce the best training images. Then, it is demonstrated that a real-time state-of-the-art object detection model can be trained to detect and differentiate CubeSats and other space targets.

INTRODUCTION

Since 1960, there have been thousands of launches of satellites from around the world, with more than 1,800 CubeSats launched by August 2022.[1] This has led to an increase in not just the number of satellites, but also the number of debris orbiting Earth. Space debris collides with other debris and space objects, creating more debris, which can further jeopardize space missions, active spacecraft, and astronauts' lives. Therefore, it is necessary to increase the situational awareness capability of deployed satellites. Furthermore, it can be useful to launch robotic satellites with actuators to automatically retrieve and remove orbiting debris. Thus, the capability to inspect space to localize and classify space targets is incredibly valuable on-board spacecraft. For all close-proximity operations between space objects, like rendezvous, docking, space debris capturing, and asteroid landing, one would hope to develop a model to eventually detect all objects in space. For a start, the authors developed a Computer Vision (CV) model to detect CubeSats (different types of CubeSats as different classes) and other random space targets (all as one class).

There has been a boom in the research of CV since 2012. Although the field has had a long history of using image processing techniques like feature (edges,[2,3] corners,[4,5] etc.) detection algorithms for downstream applications, AlexNet's performance on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC)-2012 had proven that deep Convolutional Neural Network (CNN) architecture outperforms hand-crafted features for image classification, given a large enough dataset, deep enough model, and long enough training process.[6] While the classification task only predicts the category of the image, object detection outputs the coordinates of the boxes bounding objects,

*Undergraduate Student, College of Business, University of Cincinnati, Cincinnati, OH 45221, USA.
†Assistant Professor, Department of Aerospace Engineering and Engineering Mechanics, University of Cincinnati, Cincinnati, OH 45221, USA.

their class, and their confidence scores (the probability that there is the object existing as that class at that bounding box). Soon after AlexNet, Girshick et al.[7] demonstrated that deep CNN can also outperform traditional methods of object detection. Region-based CNN (R-CNN) and its successor Fast R-CNN[8] use selective search or other object proposal algorithms to propose regions to be extracted of features for classification and bounding-box-coordinates regression. These traditional region proposal methods are time-consuming, slowing down both model inference and training (as they cannot be trained jointly). Faster R-CNN[9] replaced them with a region proposal network, which can be trained end-to-end and 10x the inference speed compared to Fast R-CNN, although it only manages to reach 17 FPS (Frames Per Second) on a K40 GPU (Graphical Processing Unit). Redmon et al.[10] introduced YOLOv1 (You-Only-Look-Once) which predicts object class and regresses location with a single-stage CNN architecture. Subsequent incremental improvements to the YOLO architecture, training and inference further proved that a simpler model can be both real-time and accurate while also being generalizable to other domains. Later, recognizing the potential from the successes of the transformers in natural language processing tasks, CV researchers have been starting to adopt this architecture in object detection with promising results for accuracy, though still falling behind YOLO architectures in terms of speed.[11,12] Such deep CNN models are extremely useful and versatile, however, there are some challenges to applying them directly to space applications.

CHALLENGES

Limited Processing Capability

While space technology has utilized cutting-edge science research, spacecraft computers have been lacking behind current State-Of-The-Art (SOTA) computation technology in terms of processing capability. Aerospace engineers have had to account for size, weight, power intake, fault tolerance, and radiation tolerance when designing an onboard information processing system. Therefore, specialized processors are needed for spacecraft computers to take advantage of real-time CV. While the latest news of NASA's High Performing Space Flight[13] project commissioning the development of a core CPU (Central Processing Unit) with 100 times[14] current space computing capability is promising, fast inference of deep CNN requires parallel computing, which on Earth can be provided by server-grade GPU or TPU (Tensor Processing Unit). For edge devices, which have some similar constraints with space applications, Google developed the Edge TPU[15] that is optimized for a high volume of low-precision neural network inference. To address space-specific problems, Goodwill et al.[16] develop a co-processor card known as the SC-LEARN (SpaceCube Low-power Edge Artificial Intelligence Resilient Node) based on the TPU. Continuous improvement to both spacecraft computers and Deep Learning (DL) will enable space applications to utilize the latest development in CV research.

Lack of Data

Supervised DL for CV required many training data, in the thousands to hundreds of thousands, to perform well on data in the wild. Labeling for object detection tasks includes drawing bounding boxes and classifying objects within an image, which takes more work than simple classification. If the labeling effort is too intensive, one can look to unsupervised, semi-supervised, or self-supervised learning methods that take advantage of unlabeled data.[17,18] Yet, like other space-related CV applications, the data for space inspection is limited. Space agencies release pictures of space objects or record CubeSats deployments for educational and marketing purposes, so most of the images are

of high quality and have few camera angles. There are plenty of videos of space and Earth from space, but few with actual man-made space objects in them, which is crucial for any non-supervised solution to space inspection. Using training data that lacks both diversity and quantity will result in a model that can easily overfit the train set. A common way to combat this lack is to synthesize more data that simulate the real images.

PREVIOUS WORKS

Previous studies have also explored ways to generate synthetic data for training space inspection CV models. Shi et al.[19] created dynamic simulations of CubeSats (1U or 3U) or servicing spacecraft with realistic orbit and altitude. Here, the camera was set to orbit for capturing an arbitrary 500 images to be trained with Faster R-CNN[9] for object detection. This approach theoretically would create foregrounds with realistic lighting and backgrounds with a logical position of the Earth in the images. It is, however, unnecessary since CV models have been shown to benefit from unrealistic objects placed within the image.[20] It can potentially harm model generalizability since the simulator limits the conditions that the model learns, i.e., the model is taught a real CubeSat can only be lit up a certain way or have certain textures. Furthermore, dynamic simulation has a huge computational overhead. The proposed work avoids these problems by diversifying space targets' textures, explicitly altering lighting conditions, utilizing a set of heuristics to randomize locations and rotations of the camera around Earth, and randomizing locations of space targets within the camera's Field Of View (FOV). Aarestad et al.[21] generated Computer-Aided Design (CAD) models of CubeSats from NASA 3D (3-Dimensional) resources on a transparent background that is randomly layered with a series of random video clips from the Internet. Then, a Mask R-CNN[22] and a YOLOv3[23] were trained on a combination of real and synthetic images for object detection. This image generation method would help decrease over-fitting when the models are trained, but it will also limit the models' capability to only detect CubeSats. This research hopes to detect CubeSats and any other foreground objects on a space background, thus one would limit to only generating space background. Samarawickrama[24] used Autodesk Maya[25] to render images of 3D models of the 1U, 2U, 3U, and 6U, before combining them with 970 web-searched images and 9,067 of the same images augmented to train a Faster R-CNN.[9] Zhang et al.[26] also used offline augmentation on web-searched data, creating a total of 24,451 images for training, instead of rendering new images from scratch. These image customizations including affine transformations or image stitching can just as easily be implemented online, augmenting the images during the training process. Aarestad et al.[21] and Shi et al.[19] rendered realistic CubeSats using detailed pre-built CAD models, a method that would consume more time than just generating different cube mesh and texturizing them during generation, which will add up to a significant difference in time consumption when synthesizing images at scale. Thus, this research took the latter approach. All these works reported above 80% mean Average Precision (mAP) on their private test sets, with private models trained on private train sets. Since a lot of the training samples and testing images were of undeployed CubeSats (in a research lab or on a table), the models are bound to perform worse with deployed CubeSats, as the lighting conditions, backgrounds, image quality, etc. are vastly different and always changing as the satellite orbits.

Instead of rendering an arbitrary number of images, with random conditions and space objects' setup, one tests these configurations via experimentation, to gain insights into how the data affect DL capabilities. As previous works used popular architectures of the past, this work also tests SOTA models for the task of space inspection, demonstrating that such models can maintain both inference

speed and accuracy while still being lightweight.

METHODOLOGY

First, a dataset of real images was collected from the Internet and labeled with bounding boxes. To fairly evaluate the CV models in the conditions in which they would be deployed, one should consider only real images of CubeSats in space to be valid for a benchmarking test set. Then, an image rendering pipeline is developed to generate images for training and testing purposes. The pipeline was implemented to have various modes of customization. Next, experimentations were conducted to test, with the same model and all else equal, which configuration of the pipeline would create the best train set. The resulting set of best configurations is used in the pipeline to produce an official dataset, on which models are trained and evaluated to produce the CV model with the highest accuracy.

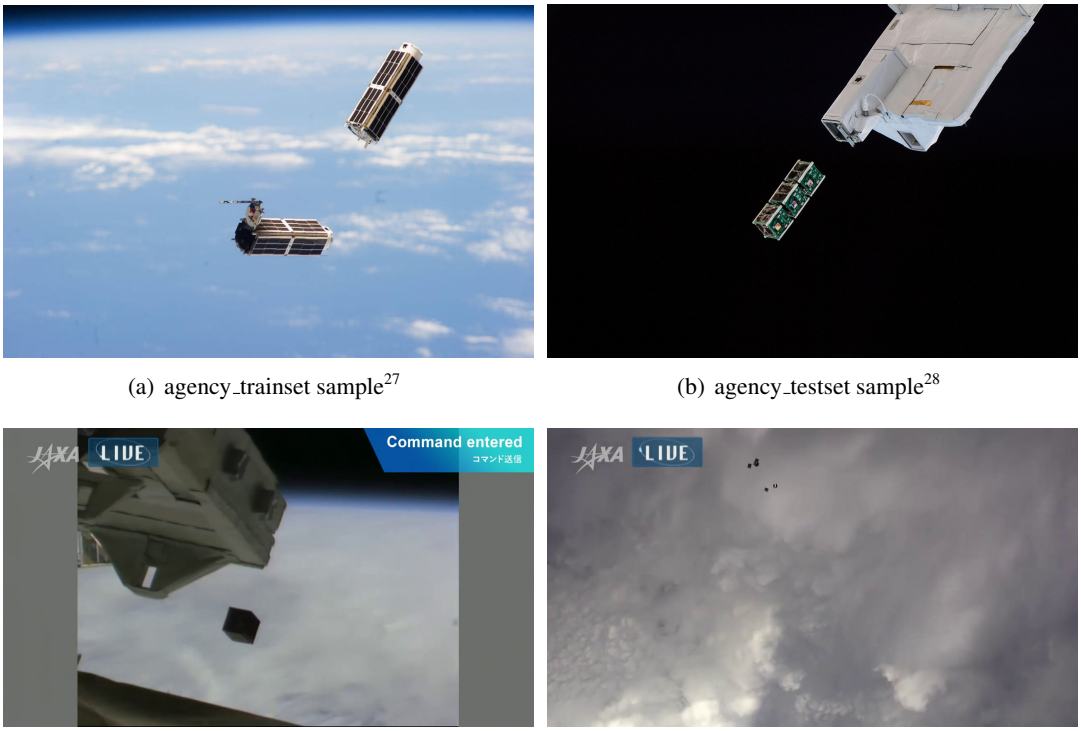
Dataset Preparation

As of December 2022, there is no public benchmark for space inspection. Since all previous works have not released their test sets to the public, it is not possible to compare their models' performances directly. Therefore, for benchmarking for future research, the authors created a dataset of images and sequences of deployed CubeSats in space, collected from various sources on the Internet and labeled with bounding boxes. The labels include different classes of CubeSats (**1U**, **2U**, **3U**, **4U**, **6U**, **12U**), Space Target (**ST**) for all other man-made space targets, and **Other** for all other unrelated parts of the image, such as light reflecting, words and logos, embedded live recording window of the control room, etc. Moons, planets, and stars, if exist, are considered part of the background. The dataset includes three test folders: agency_testset contains 37 images collected from websites of the space agency JASA and ESA, deployment_testset contains 1,014 frames from different sequences of various CubeSats being deployed from a space station, and far_testset contains 1,496 frames of camera recording CubeSats orbiting on top of a cloudy atmosphere. Sample images can be found in Figure 1. They will be used as the real test set for experiments in this research. The agency_trainset contains a set of 37 images collected from NASA that were picked for training purposes. They are diverse, yet do not exactly mirror any of the test images, to avoid both data leakage and getting a false estimation of the model's capability. The dataset also includes 288 unlabeled images from NASA and unlabeled videos from JAXA (with and without CubeSats), from which the sequence frames were extracted with various sampling rates.

Rendering Pipeline

To maximize the performance of the CV models on real data, the rendering pipeline is designed to generate images as close to reality as possible. To maximize the models' generalizability, it is also designed to create the most diverse conditions in terms of the space target's locations and rotations, camera angles, lighting intensity, etc. Such customization to the rendering pipeline can be controlled in a single configuration file for each run.

Considering the speed, ease of use and installation, and capability to produce accurate images, the authors decided to use Blender 2.82[31] and blender-python[32] to build the rendering pipeline. The pipeline uses Eevee rendering engine,[33] as it is considerably faster than Cycles[34] and other alternatives for our purposes, while still managing to create reasonably realistic images. The code repository was developed modularly so that modifications can be made to one part without affecting another.



(a) agency_trainset sample[27] (b) agency_testset sample[28]

(c) deployment_testset sample[29] (d) far_testset sample[30]

Figure 1. Sample Images Collected from the Internet for the Public Dataset

The overall architecture of the rendering pipeline is depicted in Figure 2.

Stage1. Environment Setup: A blend file is initiated or imported. The rendering process begins by randomly uniformly choosing a background mode: empty_space_partial_earth, empty_space, or full_earth. These three conditions simulate the types of background often seen in real deployed space target images. The empty_space is a pitch-black background (Figure 3(c)). The full_earth is set up so that the camera points directly towards the center of the Earth. It captures the space targets orbiting on top of Earth's atmosphere, so Earth's cloud and geography would be in the background (Figure 3(b)). The empty_space_partial_earth is a condition in between the former two, where a part of the Earth would be in the background (Figure 3(a)). These real and synthetic images of all three modes can be found in Figure 3. To save the time spent recreating Earth from scratch every time the pipeline is run, the Earth model is generated by importing a pre-built Earth model from another Blender environment, randomly layering it and rotating with different Earth and cloud textures provided by NASA.[38,39] Lighting configuration and bloom levels can be adjusted or imported from a preexisting blend. For our purpose, the light source is best set to be the Sun mode in Blender. The monocular camera is set to be the default setting in Blender 2.82.

Stage2. Space Target Generation: The space targets are either created or imported at the origin of the environment. The number of space objects is randomly chosen within a range defined in the configuration file. Space objects can be 1 of 2 types, CubeSat or ST. CubeSats objects are randomly assigned object classes 1U, 2U, 3U, 4U, 6U, or 12U, then a cube mesh of the corresponding size would be created. ST objects are imported from existing blend files containing a collection of random pre-built meshes, then to be randomly scaled to create a diversity of shapes in this class. ST

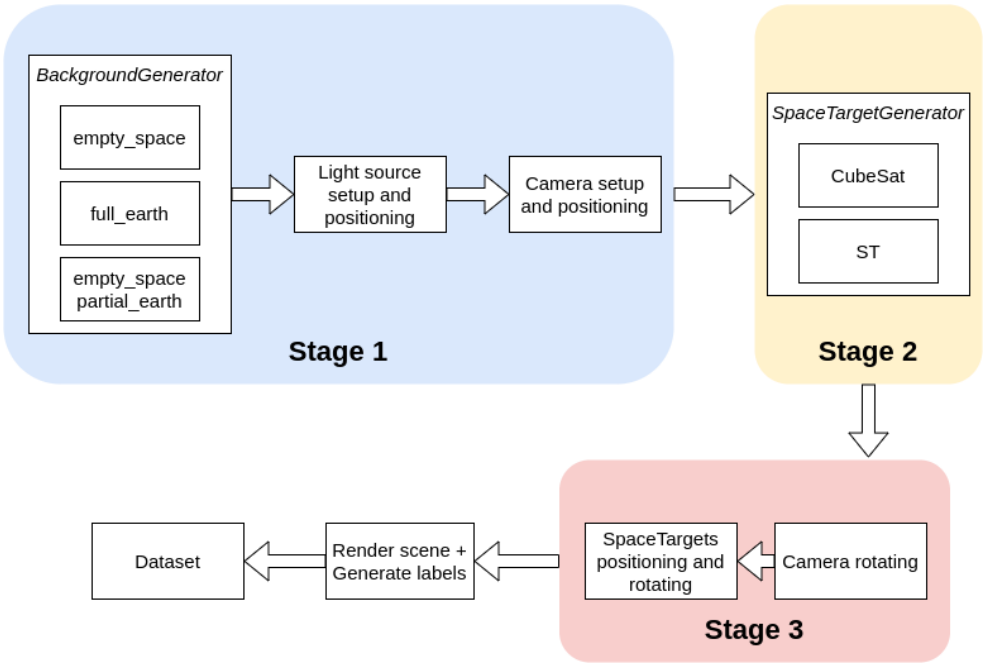
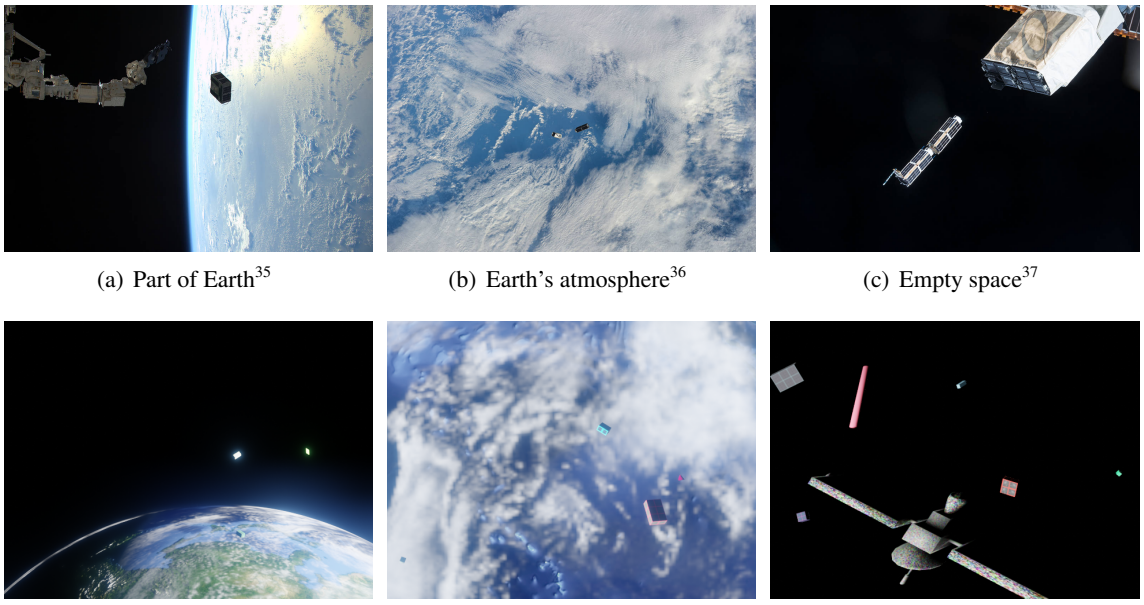


Figure 2. Blender Image Rendering Pipeline



(a) Part of Earth[35] (b) Earth's atmosphere[36] (c) Empty space[37]

(d) empty_space_partial_earth (e) full_earth (f) empty_space

Figure 3. Real Images from NASA (top) compared to Synthetic Images by the Rendering Pipeline (bottom)

objects should not contain cube-like mesh, to avoid "confusing" the model. Both will be randomly added with texture, either white noise, random solid color, or a random texture wrap collected from the Internet.

Stage3. View and Object Randomization: The camera's location would first be randomized to simulate its position orbiting Earth, with its altitude being randomized in a set tested range. For full_earth, the camera must point to the center of the Earth model, while in empty_space_partial_earth, it is rotated so that it always captures a part of Earth in its photo. For the empty_space mode, the camera is set to be in a constant location, only rolling along its z-axis. The existing space targets would then be randomly located and rotated within the FOV of the camera where their coordinates would be mapped to image coordinates and output bounding box location on the image. The image is rendered and saved to the output dataset along with its labels.

To control the rounds of image renders, the authors defined cycle, iteration, and view. For every view, the last stage is repeated. For every iteration, both stages 2 and 3 are performed. Also, for every cycle, the pipeline restarts. The frequency of each type of loop can be controlled in the configuration file.

Model Training and Evaluation

Pipeline Configuration Identification: The image rendering pipeline has a lot of configurations for customization, so it provides versatility to the data that can be generated. The common understanding in DL is that the more diverse and plentiful the training set is, the more generalizable the trained model will be. However, in some cases, this diversity comes at the cost of high computational overhead. These experiments were conducted to tune the parameters of the image rendering pipeline, while also providing insights into how data affect artificial intelligence models.

- Values of synthetic data - The authors first hoped to verify that space inspection CV models can be improved by synthetic data from the rendering pipeline. To compare the values of different training data, one needs to investigate how the same model architecture, with the same hyper-parameters, through the same training process, but trained on different data would compare to each other. Here, the pipeline rendered a synthetic train set with maximum diversity, i.e., all configurations were set to produce the most diverse images. This dataset has an arbitrary 2,000 images for training and 250 for validations. The amount was chosen to create a model that performs reasonably well, while not taking too much time to train. Three YOLOv7[40] models were trained on the synthetic dataset, on the real dataset (37 images from NASA), and on both at the same time, which will make the ratio of real to synthetic for this train set to be 1.9%. Holding all else equal, they were trained until their weights converged, and their performance on the real test sets was compared to conclude the best way to train the final model. Although the real test sets are the only valid benchmark for the space inspection task, another synthetic test set (synthetic_testset) of 4,000 images was generated to test how well the models fit the synthetic data.

- Generation mode - As the generation mode empty_space_partial_earth and full_earth would require rendering the whole Earth model, they would take much more time to generate an image compared to that of the empty_space mode. However, the empty_space mode creating only pitch-black background is further from the real data, thus theoretically would produce a worse CV model. This is also how a lot of DL research on synthetic data is set up, where the image would have an all-white empty background.[41] This assumption is tested by training a YOLOv7 model on a dataset generated by only the empty_space mode. This model would be compared to the baseline model, which was trained on data rendered with all three modes, to confirm whether the trade-off is justified.

- Number of training samples - The size of the training set places a limitation in producing an accurate and generalizable model. DL requires a huge amount of training data, but only up to a point where the model had successfully learned the distribution of the data with all its capacity and more data would provide no additional valuable information. When the extra data is noisy and faulty, it will actively harm the performance of the model. This experiment's purpose is to find the size of the final training set, a good trade-off between the amount of data, model performance, and the time it takes to train. The experimental training set will have sizes are 2,000, 5,000, 20,000, and 100,000, which is roughly the size of the Microsoft COCO (Common Objects in COntext)[42] train set. The models were trained until the loss plateaued and then evaluated on the real test set.

YOLO Object Detectors and YOLOv7: Setting it apart from previous two-stage detectors, the YOLO family of object detectors reframed the localization problem as a regression task of the bounding box coordinates, specifically as offsets of anchor boxes' coordinates. Since they are CNNs that can be trained end-to-end, the model can see the entire image during training and testing, and thus learn the contextual information of each class of objects, not just their appearance. Using parallel processing units like GPU and TPU, YOLO inference can be done extremely fast, with speed reaching 161 FPS for YOLOv7 and 286 FPS for YOLOv7-Tiny. Throughout the increment development of YOLO, improvements were made to the model architecture, training processes, label assignment, and loss function, such as using feature maps of different scales for detection to detect small objects,[23] utilizing bags of freebies like data augmentation and bags of specials like different regression objective functions,[43] model scaling for different devices,[44] etc.

YOLOv7 is the latest and current SOTA object detection model for real-time object detection. It surpasses all past YOLO models and other contemporary object detectors like DN-Deformable DETR[45] and DINO-5scale-R50[46] in terms of both speed and accuracy while having a smaller number of parameters. YOLOv7 improved upon the previous work by Wang et al.[47] to introduce extended efficient layer aggregation networks as the backbone of the model, which can learn more diverse features while optimizing the use of parameters and computations. It also introduced compound scaling for concatenation-based models and proposed a new method to re-parameterize convolutional blocks. While Wang et al.[40] also introduced YOLOv7-Tiny with very few parameters for edge devices, they did not release the weights trained on the COCO dataset, and there was a huge drop in Average Precision (AP) on COCO test set compared to YOLOv7. Expecting the pre-trained weights on COCO to be beneficial to training with little data, this research would use the YOLOv7 architecture, considering it a good trade-off between model size, training speed, inference speed, and accuracy.

Domain Adaptation: Since this research looks to train on synthetic data and test on real data, the problem of difference in data distribution can arise. In CV, this domain shift can be caused by image degradation, different lighting condition, rendered images not being able to completely simulate reality, etc. The train data and test data usually do not look similar enough that the trained model could not generalize on the test set or in the deployment environment, and thus it needs to be trained with Domain Adaptation (DA). DA seeks to align the distribution of the data from the source domain and data from the target domain. DA methods look to extract domain-invariant representations from the data and bridge the gap between the source and target data distributions, so its effectiveness will be dependent on the discrepancy between the two domains, i.e., how different the two data are from each other. Therefore, this research hopes to start by experimenting with the effectiveness of some discrepancy-based approaches on this research's synthetic data and real test set. Most research in

DA focuses on classification, so the methods that are the most easily transferable to the object detection tasks are the statistical criterion. These methods align the statistical distribution shift between the feature maps by adding a loss function to guide the learning of the model's backbone. Popular statistical approaches include Maximum-Mean Discrepancy (MMD) loss,[48,49] Kullback-Leibler (KL) divergence,[50] etc. Sun and Saenko[51] introduced the deep CORAL (CORrelation ALignment for deep DA) loss that aligns source and target correlation by matching the covariance matrices of feature maps extracted from both domains. Zellinger et al.[52] expanded on MMD and KL divergence by introducing Central Moment Discrepancy (CMD) loss, which matches the higher-order moments of the two domains' hidden features. These two losses were shown to have outperformed other functions while being competitive with each other. In both the CORAL and CMD papers, these losses were applied to their classification models' last fully connected layers' output. To stay true to the purpose of guiding the feature extractor to learn domain-invariant features, in YOLOv7, these loss functions were applied to the backbone of the model. Because neural networks learned features at various levels of abstraction at the varying depths of the model, the different positions on which the losses were applied would affect the domain alignment differently. Firstly, these losses were experimented with on the final output of the backbone before the head of the model (named L1). Secondly, they were applied to the three hidden activations in the model's backbone that were output to be concatenated with feature maps in the model's head (named L3). Lastly, they were applied to the feature maps before the three final detection heads (named L3d). In all cases, the gradients should be back-propagated throughout the backbone from the loss-applied layers. The unlabeled data from this paper's public dataset would be used as target data to be trained along with the synthetic and real train sets.

Evaluation Metrics: As it is standard in the field of object detection, this research uses the mAP as the main evaluation metric. Detection models output a confidence score with every prediction bounding box. That is a probability that indicates how likely an object exists at that box location as that class. The detection is only considered valid if its confidence score reaches over a confidence threshold.

A prediction bounding box (Figure 4) is considered a positive detection when the IoU (Intersection over Union) of the prediction box (B_p) and label/ground truth box (B_t) is over a certain threshold. IoU is calculated as follows:

$$\text{IoU} = \frac{\text{area}(B_p \cap B_t)}{\text{area}(B_p \cup B_t)} \tag{1}$$

In object detection, a prediction is considered a True Positive (TP), when it is a positive detection, and it correctly classifies the object. Conversely, a False Positive (FP) is a misclassified positive detection or part of the background detected as an object. A False Negative (FN) is an undetected object. Here, precision (p) and recall (r) for each class can be defined as follows:

$$p = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad \text{and} \quad r = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2}$$

As the confidence threshold decreases, more predictions become valid, and presumably TP increases and FN decreases. Therefore, there is a trade-off between p and r as the confidence threshold changes. The precision-recall curve is the plot of p (y-axis) and r (x-axis) values for different confidence thresholds from 0 to 1. The AP at IoU threshold x (AP@x) is defined as the area under the curve, which indicates the mean accuracy of the model across varying thresholds of confidence:[53]

$$\text{AP@}x = \int_0^1 p(r)\text{d}r \tag{3}$$

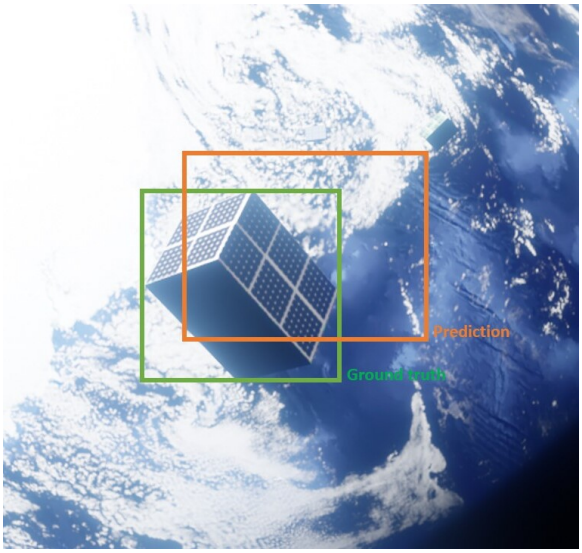


Figure 4. Prediction vs Ground Truth Bounding Boxes

By extension, the mAP at IoU threshold x (mAP@x) is the average of the AP@x metrics of all classes as follows:[53]

$$\text{mAP@}x = \frac{1}{C}\sum_{i=1}^{C}\text{AP}_i@x \tag{4}$$

where C is total number of classes, and $\text{AP}_i@x$ is the AP@x value of the i-th class.

By the definition of the COCO dataset's evaluation, mAP@.5:.95 is the mAP values taken average over IoU thresholds from 0.5 to 0.95 with intervals of 0.05.

RESULTS AND DISCUSSION

Public Dataset

Most of the public dataset only has only 1U, 3U, or ST objects (Figure 5).

ST objects commonly found are parts of the International Space Station, including the solar panel and the Nanoracks CubeSat Deployer. The only instances of Other are the JAXA's logos on screen. As the deployment_testset and far_testset are extracted from JAXA's videos, the logos exist in every frame.

Since there are fewer than 10 instances of 2U, 4U, 6U, and 12U objects for each real test set (as shown in Figure 5, the model cannot be evaluated fairly on its ability to detect these classes in deployment. The model also cannot perform well on the class Other, because there is no instance of this class in any train set. Therefore, model performance evaluation will only account for 1U, 3U, and ST.

When labeling the data, some challenges arose from the way this problem was formulated, from which one can make observations about the nature of the problem. Firstly, all CubeSats can look the same from certain angles. For example, there is no way to differentiate between 1U, 2U, or 3U CubeSats roughly facing their single-square side towards the camera (Figure 6(a)). Secondly, multiple CubeSats can look like one from certain angles, especially when three 1U CubeSats are

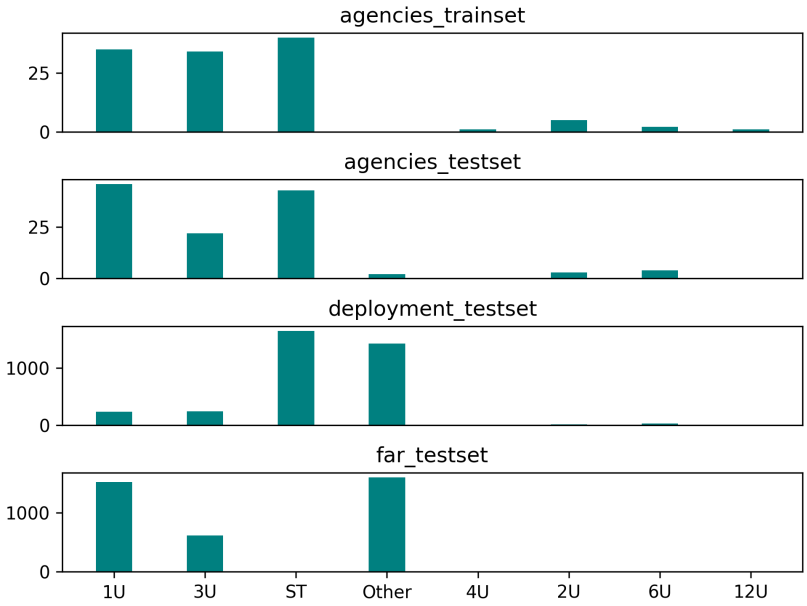
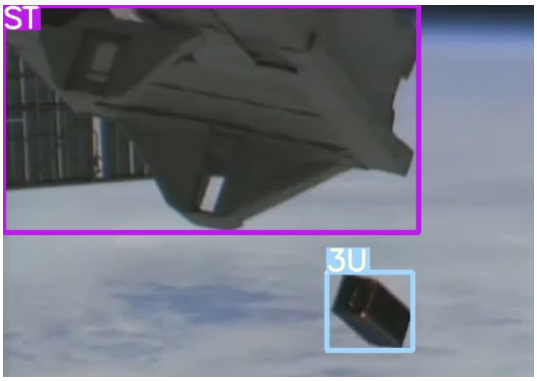
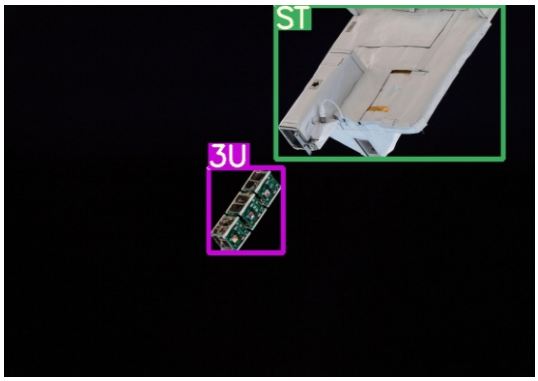


Figure 5. Number of Objects per Each Dataset

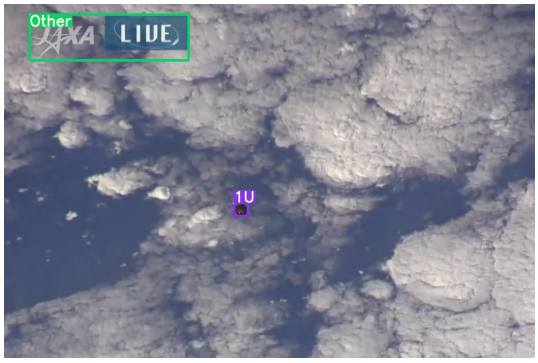


(a) CubeSats that could either be 3U or 2U[54]



(b) 3 1Us look visually like 1 3U[28]

(c) Complicated ST[55]



(d) The JAXA Logo is considered Other[56]

Figure 6. Problems with Labeling the Dataset

Table 1. Rendering Speed Benchmarking on GeForce RTX 3070 Ti GPU

Generation mode	empty_space	full_earth	All modes
Rendering speed (sec/image)	0.56	5.97	3.84

released, they are often stacked next to each other, which would sometimes confuse even humans using only visuals to classify the object (Figure 6(b)). Thirdly, this research defined ST as any other man-made space object in space, so it can be complicated to decide where the box of one object ends and the next one begins, or some bounding box would contain the whole image because the ST is big and weirdly shaped (Figure 6(c)). Lastly, the way this research defined the Other object class makes it so that there is too much variety in the appearance of Other, thus condemning this class to never be well simulated, and thus on which the model can never perform well (Figure 6(d)).

The first two issues concern the confusion differentiating between CubeSats for both the human eyes and CV models. They cannot be resolved on purely the visual of a single frame, and so there would be both bad data that were used to train the models (an appearance of a single square is sometimes labeled 1U and sometimes 3U, an appearance of a 10 cm by 10 cm by 30 cm CubeSat are sometimes labeled one 3U and sometimes three 1Us) and confusing data that is impossible to predict accurately. This research controlled for that as much as possible, by labeling the data the way it looks, not what it actually is with the knowledge of its context (a stack of three 1Us closed enough to each other is labeled 3U). It is also theoretically possible to implement the same rules in the rendering pipeline. However, the problem remains and would put a cap on how well the model performs. In CV, the task of object tracking[57] includes detecting an object, tagging it with an ID, and tracking that object across time. Using these object tracking methods, one can potentially smooth out these classification errors, i.e., as a 3U CubeSats rotates, the detection model may classify it as a 1U from certain angles, but tracking can correct it as the same 3U object. While this solution only works on sequences and is not applicable to this research's scope, it will be beneficial for the deployment of a space inspection system. The latter two issues concern bounding boxes and label classification of the vast variety of man-made objects in space and other oddities. One can consider removing bounding boxes altogether and reformulating this problem. It could be seen as a background-foreground segmentation problem, which would lose any object class information. It can also be reformulated as an instance segmentation problem, which would still run into the issue of deciding where an instance ends, and another begins with just vision. Each reformulation of this problem would depend on the requirements of the downstream application, i.e., object grasping would require the granularity of segmentation's pixel output, while depth estimation using a fusion of sensors and cameras may only need bounding box output.

Rendering Pipeline

The rendering speed of the rendering pipeline is recorded by generating 100 images (25 cycles, 2 iterations per cycle, 2 views per iteration), shown in Table 1. For both the full_earth and empty_space_partial_earth modes, the pipeline needs to render Earth, so it should have the same average rendering speed.

Table 2. Models' Performance (mAP@.5) Evaluation on 1U, 3U, and ST based on Data Generation Configuration

Training data	No.	agency_testset	deployment_testset	far_testset	synth_testset
Real (Baseline)	37	0.72	0.47	0.44	0.07
Synthetic	2000	0.29	0.27	0.41	0.47
RS2000	2037	**0.73** (+1%)	**0.54** (+15%)	**0.6** (+38%)	0.63
empty_space	2037	0.44	0.3	0.45	0.36
RS5000	5111	0.71	0.52	0.56	**0.75**
RS2000_tuned	2037	0.67	0.42	0.53	0.64

Model Training and Evaluation

Pipeline Configuration Identification: Before creating an official model, an official train set needs to be produced from the best configuration of the rendering pipeline.

- Value of Synthetic Data - YOLOv7 models were set to train on 37 real images (from now on will be referred to as baseline model and dataset), 2,000 synthetic images, and a combination of the two datasets. They used the same set of algorithmically evolved hyper-parameters used to train the COCO dataset. As they were trained until the models are in convergence, one found that training with a combination of both sets yields the highest performing model (Table 2) on all real test sets.

  This model and dataset will be referred to as **RS2000**. Given that the agency_testset consists of miscellaneous deployment images and high-quality photos, a lot like the agency_trainset, it's expected that both the baseline model and the RS2000 model perform well. However, RS2000 is a lot more accurate on the far_testset, suggesting that the synthetic data contains small CubeSats samples that the agency_trainset lacks. This increase in mAP overall showed that the synthetic data was helpful in generalizing the model, while the real data, no matter how little, provided crucial information to the model.

- Generation Mode - Another YOLOv7 model was trained on data with only empty space synthetic data and real data, but this did not perform as well as the RS2000. One can conclude that the variety of background textures in the data is indeed useful to the model's learning.

- Number of Training Samples - A training set of over 5,000 images (RS500), including 2,000 synthetic images used for the RS2000, 3,000 additional synthetic images, and 37 real images that were sampled three times each to maintain the same ratio of real-to-synthetic of the RS2000 data, was used to train a YOLOv7. However, RS5000 performed worse compared to RS2000 by a few percentages. This could mean that RS5000 overfitted to the real training samples, but more likely it overfitted to the synthetic data, shown by its better performance on the synthetic test set compared RS2000. The authors trained and tuned another model (RS2000_tuned) on the original RS2000 data, by adding training image weights, training on multiple scales of the input, evolving hyper-parameters, and adding more data augmentation. This only increased the model's accuracy on the synthetic data, while losing some performance on the real data. The synthetic data is inherently different from the real data, so more synthetic training data would only help the model fit this synthetic data domain. Therefore, DA was used to reduce the domain shift.

Table 3. Models' Performance (mAP@.5) Evaluation on full_real_testset with 1U, 3U, and ST

Model	Real			Synthetic		
mAP@.5	1U	3U	ST	1U	3U	ST
RS2000	0.51	**0.68**	**0.47**	0.65	0.66	0.57
CORAL (L1)	**0.55** (+8%)	0.57	0.09	0.68	0.71	0.56
CORAL (L3)	0.46	0.43	0.17	0.68	0.71	0.57
CORAL (L3d)	0.5	0.59	0.43	0.67	0.68	0.57
CMD (L1)	0.51	0.54	0.15	0.67	0.71	0.57
CMD (L3)	0.42	0.6	0.11	**0.69**	**0.73**	**0.57**
CMD (L3d)	0.45	0.62	0.38	0.69	0.68	0.54

Table 4. Models' mAP with full_real_testset on 1U, 3U, and ST

@IoU	Obj size	baseline	RS2000
0.5:0.95	all	0.37	0.39
0.50	all	0.49	0.55
0.75	all	0.43	0.46
0.5:0.95	small	0.25	0.44
0.5:0.95	medium	0.35	0.38
0.5:0.95	large	0.42	0.51

True to the few-shot object detection fine-tuning paradigm,[17] the authors also finetuned the model with the COCO train set along with a dataset of the novel space inspection object classes, first the real train set and then the RS2000 dataset, finding that in both cases the models do not predict any true positive for CubeSats or ST while predicting COCO base classes with great accuracy.

To close the domain shift between the real and synthetic data distributions, models were trained with the CORAL and CMD domain adaptation loss on the RS2000 train set. They were evaluated on the combination of all three real test sets, called full_real_testset. Their accuracy (shown in Table 3) degraded on the synthetic test set compared to RS2000's. As expected, they are no longer overfitting to the synthetic data. To validate the models' performance in the deployment environment, however, they need to perform well on the real test sets. Unfortunately, they did not outperform the RS2000 model with all the object classes. Only the mAP of CMD (L1) managed to match that of the RS2000 model on the real test sets with 1U CubeSats, but it failed to also outperform with 3U and ST. Interestingly, the authors also observed that the unlabeled data did guide the models to learn to extract features that can generalize better to the synthetic data. The DA losses had reached their global minimum by the end of the training processes; therefore, the losses have effectively aligned the statistical distribution shift between the source and the target feature maps. One can conclude that statistical approaches to DA are not appropriate for YOLOv7 in this domain of data.

The RS2000 model trained purely on a combination of synthetic and real data remained the most accurate model. In Table 4, one can examine the RS2000's mAP across IoU thresholds and object size compared with the baseline model trained using only real data. Overall, RS2000 outperforms the baseline model on all scales of space objects. The RS2000's mAP suffers a light decrease when predicting objects of medium size, suggesting that with multi-scale training or modification to the training data, the model can become more robust to objects' scales. RS2000's accuracy still leaves much to be desired. Limited by the model size and the lack of real data, both labeled

and unlabeled, this work should be further improved by finding a DA approach that can utilize the large amount of data that can be generated by the rendering pipeline. While most research on domain adaptive object detection is with Faster R-CNN, some have found successes with cross-domain one-stage detectors, adapting camera quality, clear-to-foggy weather,[58] light-to-dark,[59] and synthetic-to-real street images.[60] Integrating a domain adaptive network into YOLOv7 or using generative adversarial training can potentially reduce the gap between the synthetic and real data domain, thereby improving performance on the test sets.

CONCLUSION

This research introduces a public dataset of images of CubeSats, labeled or unlabeled, for supervised and un/semi-supervised learning tasks for space inspection. It also concludes that there are errors inherent to the problem's nature, which would limit a Computer Vision (CV) system's capability. The authors designed a rendering pipeline to generate data diverse enough to produce an accurate CV model, then demonstrated that YOLOv7 can be trained to detect CubeSats and other space targets in space. When met with the issue of domain shift between the synthetic data and real data, this research applied domain adaptation using objective functions that affect statistical metrics of hidden representation. They, however, did not increase the generalizability of the model to the real data, thus further research is needed to find an approach that works well with this data domain.

REFERENCES

[1] "Nanosats Database," `https://www.nanosats.eu/`. Retrieved: 2022-12-18.

[2] J. Canny, "A Computational Approach to Edge Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-8, No. 6, 1986, pp. 679–698, 10.1109/TPAMI.1986.4767851.

[3] I. Sobel, "An Isotropic 3x3 Image Gradient Operator," *Presentation at Stanford A.I. Project 1968*, 02 2014.

[4] J. Shi and C. Tomasi, "Good features to track," *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 1994, pp. 593–600, 10.1109/CVPR.1994.323794.

[5] C. Harris and M. Stephens, "A Combined Corner and Edge Detector," *Alvey Vision Conference*, Vol. 15, 02 1988.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, Red Hook, NY, USA, Curran Associates Inc., 2012, p. 1097–1105.

[7] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Region-Based Convolutional Networks for Accurate Object Detection and Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 38, No. 1, 2016, pp. 142–158, 10.1109/TPAMI.2015.2437384.

[8] R. Girshick, "Fast R-CNN," *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1440–1448, 10.1109/ICCV.2015.169.

[9] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 39, No. 6, 2017, pp. 1137–1149, 10.1109/TPAMI.2016.2577031.

[10] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2015, 10.48550/ARXIV.1506.02640.

[11] A. Kolesnikov, A. Dosovitskiy, D. Weissenborn, G. Heigold, J. Uszkoreit, L. Beyer, M. Minderer, M. Dehghani, N. Houlsby, S. Gelly, T. Unterthiner, and X. Zhai, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," 2021.

[12] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-End Object Detection with Transformers," *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I*, Springer-Verlag, 2020, p. 213–229, 10.1007/978-3-030-58452-8_13.

[13] "High Performance Spaceflight Computing (HPSC)," `https://www.nasa.gov/directorates/spacetech/game_changing_development/projects/HPSC`. Accessed: 2023-04-01.

[14] "NASA Makes RISC-V the Go-to Ecosystem for Future Space Missions," https://www.sifive.com/press/nasa-selects-sifive-and-makes-risc-v-the-go-to-ecosystem. Accessed: 2023-04-01.

[15] "Coral," https://coral.ai/. Accessed: 2023-04-01.

[16] J. Goodwill, G. Crum, J. MacKinnon, C. Brewer, M. Monaghan, T. Wise, and C. Wilson, "NASA SpaceCube Edge TPU SmallSat Card for Autonomous Operations and Onboard Science-Data Analysis," *Small Satellite Conference 2021 Proceedings*, 2021.

[17] G. Huang, I. Laradji, D. Vazquez, S. Lacoste-Julien, and P. Rodriguez, "A Survey of Self-Supervised and Few-Shot Object Detection," 2021, 10.48550/ARXIV.2110.14711.

[18] Y. Chen, M. Mancini, X. Zhu, and Z. Akata, "Semi-Supervised and Unsupervised Deep Visual Learning: A Survey," 2022, 10.48550/ARXIV.2208.11296.

[19] J.-F. Shi, S. Ulrich, and S. Ruel, *CubeSat Simulation and Detection using Monocular Camera Images and Convolutional Neural Networks*, 10.2514/6.2018-1604.

[20] G. Ghiasi, Y. Cui, A. Srinivas, R. Qian, T.-Y. Lin, E. D. Cubuk, Q. V. Le, and B. Zoph, "Simple Copy-Paste is a Strong Data Augmentation Method for Instance Segmentation," *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 2917–2927, 10.1109/CVPR46437.2021.00294.

[21] J. Aarestad, A. Cochrane, M. Hannon, E. Kain, C. Kief, S. Lindsley, and B. Zufelt, "Challenges and Opportunities for CubeSat Detection for Space Situational Awareness using a CNN," 2020.

[22] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 2980–2988, 10.1109/ICCV.2017.322.

[23] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," *ArXiv*, Vol. abs/1804.02767, 2018.

[24] S. N. G. I. Samarawickrama, *Faster R-CNN based CubeSat close proximity detection and attitude estimation in the Department of Aerospace Engineering*. 2019.

[25] Autodesk, INC., "Maya,"

[26] C. Zhang, B. Guo, N. Liao, Q. Zhong, H. Liu, C. Li, and J. Gong, *STAR-24K: A Public Dataset for Space Common Target Detection*, Vol. 16. 2 2022.

[27] "File:ISS-38 NanoRacks CubeSat Deployment Iss038e044916.jpg," https://commons.wikimedia.org/wiki/File:ISS-38_NanoRacks_CubeSat_Deployment_Iss038e044916.jpg. Retrieved: 2022-12-24.

[28] "Birds-2 deployed from Kibo (Iss056e130478).jpg," https://commons.wikimedia.org/wiki/File:Birds-2_deployed_from_Kibo_(Iss056e130478).jpg. Retrieved: 2022-12-24.

[29] "Small Satellites Deployment J-SSOD no16 from "Kibo" (OPUSAT-II, BIRDS-4, RSP01,WARP01, timestamp 1:05:49)," https://www.youtube.com/watch?v=Yt26Z1EJIDc. Retrieved: 2022-12-24.

[30] "Small Satellites Deployment J-SSOD no16 from "Kibo" (OPUSAT-II, BIRDS-4, RSP01,WARP01, timestamp 38:19," https://www.youtube.com/watch?v=Yt26Z1EJIDc. Retrieved: 2022-12-24.

[31] "Blender 2.82 Release Notes," https://wiki.blender.org/wiki/Reference/Release_Notes/2.82. Retrieved: 2022-12-20.

[32] "Blender Python API Documentation," https://docs.blender.org/api/current/index.html. Retrieved: 2022-12-20.

[33] "Eevee Rendering Engine Documentation," https://docs.blender.org/manual/en/latest/render/eevee/introduction.html. Retrieved: 2022-12-20.

[34] "Cycles Rendering Engine Documentation," https://docs.blender.org/manual/en/latest/render/cycles/introduction.html. Retrieved: 2022-12-20.

[35] "4U CubeSat deployed in space, originally found on NASA CubeSat Image Gallery," https://www.flickr.com/photos/162156874@N06/46467991585/. Retrieved: 2022-12-26.

[36] "A set of NanoRacks CubeSats is photographed by an Expedition 38 crew member after deployment," https://www.flickr.com/photos/nasa2explore/12801808603/. Retrieved: 2022-12-26.

[37] "On Feb. 27 2015, a series of CubeSats, small experimental satellites, were deployed via a special device mounted on the Japanese Experiment Module (JEM) Remote Manipulator System (JEMRMS)," https://www.nasa.gov/content/cubesat-deployment. Retrieved: 2022-12-26.

[38] "NASA's Catalog of images of Earth textures," https://visibleearth.nasa.gov/collection/1484/blue-marble. Retrieved: 2022-12-21.

[39] "NASA's Catalog of images of Cloud on Earth," https://www.visibleearth.nasa.gov/images/57747/blue-marble-clouds. Retrieved: 2022-12-21.

[40] C.-Y. Wang, A. Bochkovskiy, and H.-y. Liao, "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," 07 2022, 10.48550/arXiv.2207.02696.

[41] X. Peng, B. Usman, K. Saito, N. Kaushik, J. Hoffman, and K. Saenko, "Syn2Real: A New Benchmark forSynthetic-to-Real Visual Domain Adaptation," *ArXiv*, Vol. abs/1806.09755, 2018.

[42] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common Objects in Context," *Computer Vision – ECCV 2014* (D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, eds.), Cham, Springer International Publishing, 2014, pp. 740–755.

[43] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," *ArXiv*, Vol. abs/2004.10934, 2020.

[44] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Scaled-YOLOv4: Scaling Cross Stage Partial Network," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021, pp. 13029–13038.

[45] F. Li, H. Zhang, S. Liu, J. Guo, L. M. Ni, and L. Zhang, "Dn-detr: Accelerate detr training by introducing query denoising," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 13619–13627.

[46] H. Zhang, F. Li, S. Liu, L. Zhang, H. Su, J. Zhu, L. M. Ni, and H.-Y. Shum, "DINO: DETR with Improved DeNoising Anchor Boxes for End-to-End Object Detection," 2022.

[47] C.-Y. Wang, H.-Y. M. Liao, and I.-H. Yeh, "Designing Network Design Strategies Through Gradient Path Analysis," 2022, 10.48550/ARXIV.2211.04800.

[48] H. Yan, Y. Ding, P. Li, Q. Wang, Y. Xu, and W. Zuo, "Mind the Class Weight Bias: Weighted Maximum Mean Discrepancy for Unsupervised Domain Adaptation," 05 2017.

[49] E. Tzeng, J. Hoffman, T. Darrell, and K. Saenko, "Simultaneous Deep Transfer Across Domains and Tasks," 12 2015, pp. 4068–4076, 10.1109/ICCV.2015.463.

[50] F. Zhuang, X. Cheng, P. Luo, S. J. Pan, and Q. He, "Supervised Representation Learning: Transfer Learning with Deep Autoencoders," *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, AAAI Press, 2015, p. 4119–4125.

[51] B. Sun and K. Saenko, "Deep CORAL: Correlation Alignment for Deep Domain Adaptation," *Computer Vision – ECCV 2016 Workshops* (G. Hua and H. Jégou, eds.), Cham, Springer International Publishing, 2016, pp. 443–450.

[52] W. Zellinger, T. Grubinger, E. Lughofer, T. Natschläger, and S. Saminger-Platz, "Central Moment Discrepancy (CMD) for Domain-Invariant Representation Learning," 2017, 10.48550/ARXIV.1702.08811.

[53] R. Padilla, W. L. Passos, T. L. B. Dias, S. L. Netto, and E. A. B. d. Silva, "A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit," *Electronics*, Vol. 10, Jan 2021, p. 279, 10.3390/electronics10030279.

[54] "Three CubeSats successfully deployed from "Kibo"!," https://iss.jaxa.jp/en/kiboexp/news/1810_cubesat10_en.html. Retrieved: 2023-1-1.

[55] "JAXA and UNISEC open of J-CUBE Program!," https://humans-in-space.jaxa.jp/en/biz-lab/news/detail/001787.html. Retrieved: 2023-1-1.

[56] "Small Satellites Deployment J-SSOD no16 from "Kibo" (OPUSAT-II, BIRDS-4, RSP01,WARP01, timestamp 1:07:22)," https://www.youtube.com/watch?v=Yt26Z1EJIDc. Retrieved: 2022-12-24.

[57] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: A survey.," *ACM Comput. Surv.*, Vol. 38, No. 4, 2006, p. 13.

[58] S. Zhang, H. Tuo, J. Hu, and Z. Jing, "Domain Adaptive YOLO for One-Stage Cross-Domain Detection," 2021, 10.48550/ARXIV.2106.13939.

[59] Y. Sasagawa and H. Nagahara, "YOLO in the Dark - Domain Adaptation Method for Merging Multiple Models," *Computer Vision – ECCV 2020* (A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, eds.), Cham, Springer International Publishing, 2020, pp. 345–359.

[60] M. Hnewa and H. Radha, "Multiscale Domain Adaptive YOLO for Cross-Domain Object Detection," *CoRR*, Vol. abs/2106.01483, 2021.