Training Neural Networks to Detect Resident Space Objects using Space Based Optical Payloads and Low-SWaP Onboard Processing

Dominique Low, Christos Koulas, Domenico Di Giovanni
MDA Systems

ABSTRACT

The amount of space debris and satellites in orbit is increasing, making the on-board detection of Resident Space Objects (RSOs) evermore important for spacecraft situational awareness and collision mitigation. On-board low-cost optical sensors are increasingly being used to produce image data for Space Situational Awareness (SSA). The ability to accurately interpret optical sensor image data for RSOs and propagate their orbits rapidly is a fundamental building block for applications like automated flight dynamics for collision avoidance.

The RSO Detector Convolution Neural Network (CNN) is a notable detection model developed by MDA with support of the Canadian Space Agency (CSA). The model extracts the position information for RSOs in optical images of star streaks and was trained with simulated images based on Sapphire's principal payload. Verification of the model was based on the proximity of predicted RSOs to the position of the truth mask RSOs. The model was able to achieve 83% accuracy with fully simulated on-orbit data. Two-thirds of the detection errors are attributed to missed RSOs and one third attributed to false positives. Detection performance improved when reducing features associated with non-nominal image data, suggesting that broader on-orbit optical image datasets and greater temporal coverage would improve model accuracy. The RSO Detector CNN was then run on a configurable Graphical Processing Unit (GPU) in an effort to mimic a space qualified System on Chip (SoC), making RSO location predictions with an average speed of 23ms.

The success of the RSO Detector CNN demonstrates that this technique would be suitable for closed loop applications such as automated flight control and other time-critical operations. This capability will prove to be key when integrated with an onboard SSA system.

1. INTRODUCTION

Detecting and tracking RSOs with on-orbit payloads is increasingly crucial for the needs of future spacecraft navigation. Conventional methods are slow and often require a human operator to analyze downlinked or external data for SSA. As the quantity of objects in orbit increase, so do does the risks from the decision support systems used to avoid unplanned conjunctions. Shortcomings with conventional SSA mean unanticipated changes in RSO behaviour, such as evasive maneuvering, will occur frequently and reaction times to such situations must be handled more efficiently.

Advancements in Machine Learning (ML) enable spacecraft systems to perform their own detection and evaluation of collision hazards using onboard sensors. Rather than relying on ground-based manual evaluation of third-party data sources, a satellite can perform its own collision hazard identification and mitigation without human involvement – saving limited time and reducing cost.

To address the challenges of SSA in crowded orbits, we created a ML model trained to detect the location of RSOs among a field of stars using optical imagery. This has been achieved in three parts: the creation of an image simulator to generate large datasets of simulated satellite imagery, the creation of a ML modified U-Net model, and the deployment and optimization of the trained model on a low power GPU.

The ability to simulate heterogeneous sensor types or non-nominal image data was desired, and thus a fully configurable image simulator was created. The user of the image simulator can configure attributes such as the Field of View (FOV) and the image size to simulate a particular camera. For this project, the simulator was configured to create synthetic imagery representative of the DND Sapphire Mission, where actual payload data was available for verification. Additionally, the image simulator allows one to specify attributes in the image, featuring various types of noise, dead pixels, and other common attributes found in on-orbit data.

Once trained, the modified U-Net model detected the location of RSOs among a field of stars with an 83% accuracy. The trained model was then deployed onto the Nvidia Xavier Jetson GPU. We optimized throughput, power, efficiency, and inference time. We experimented with different inference engines and can successfully predict the location of RSOs from optical data image within 23ms.

In this report we outline the Image Simulator, the RSO detection CNN, and the design decisions made during the capability trade-off assessment. We present the simulated images in comparison to Sapphire image data, the predictions of our ML model, and the final performance metrics achieved on a low power GPU. Furthermore, we discuss the use of this CNN in a broader SSA system which leverages this capability for an on-orbit decision making algorithm.

2. PROTOTYPE DEVELOPMENT

2.1 OVERVIEW

Through the SSA OBP project, we have created a ML algorithm to detect RSOs among a field of stars. This model was trained on custom generated simulated images. Features were added iteratively to the images, allowing us to tune datasets with the desired features, ensuring their likeness to real imagery.

The structure of the model was inspired from the Deeptrack 2.0 U-Net model [1]. Adjustments and optimizations to the original CNN's construction were made. Various datasets were used as training inputs, including various epochs, batch sizes, penalties, and loss functions. Once satisfied with the accuracy results, the model was run on the GPU, and the resulting performance data was recorded.

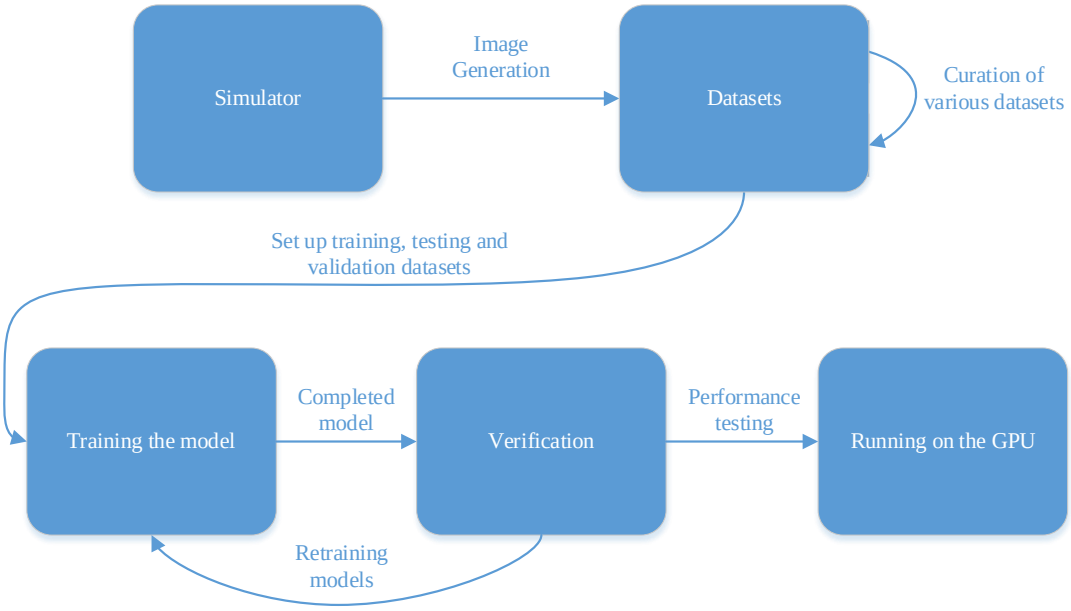

Figure 4-1 Work Cycle of the RSO Detector CNN

2.2 IMAGE SIMULATOR

The image simulator was a necessary component of this project due to the limited amount of source imagery data. This image simulator produces images akin to the view from the imaging satellite, complete with the presence of RSOs on a background of photorealistic stars. For our purposes, we used a selection of Geostationary Earth Orbit (GEO) satellites to serve as RSO targets. All orbits were propagated leveraging the AstroPy and Skyfield Python libraries. The simulator provides substantial flexibility in the curation of the datasets, allowing us to customize for edge cases, different sensor types, different image satellites, etc. Because the physics of orbital mechanics is very

well understood and our synthesized images are very similar to those taken by real satellites, we have successfully trained an algorithm using this imagery.

2.2.1 METHODOLOGY

Figure 4-2 shows the Simulator's order of operations.

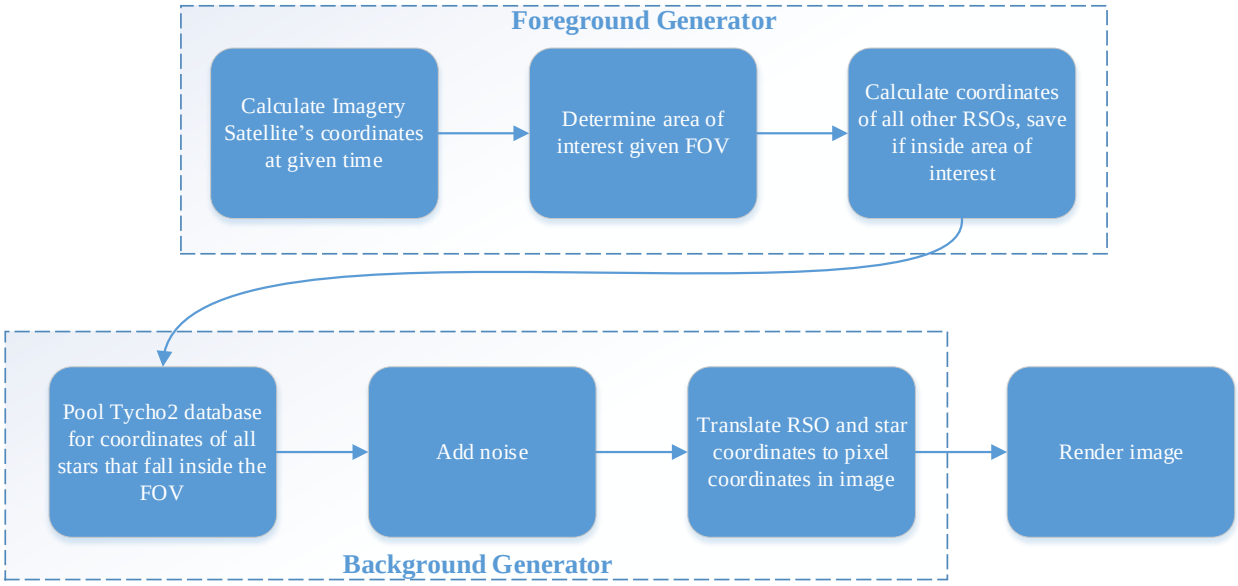

Figure 4-2 Simulator Order of Operations

2.2.2 FOREGROUND GENERATOR

The Foreground Generator uses accurate orbital information to project all satellites into their correct position in the frame of the image. Using the SGP4 propagation method, we leverage each satellite's classical orbital elements to determine its coordinates given a timestamp. First, the simulator propagates the imagery satellite, and uses the configurable FOV angle attribute to determine the area of the sky that the satellite would be able to "see" at the given time. Next, all other initialized satellites are propagated to determine their respective Right Ascension (RA) and Declination (Dec) coordinates. Should these background satellites fall in the FOV of the image satellite, their coordinates will be input into the rendered image. To create the illusion of varying brightness in RSOs and stars due to distance, relative light emission levels, reflective and refractive properties, we implemented a configurable scale of brightness that incorporates a pseudo random element and generates a realistic brightness variation across the entire image. In this project, we selected the Anik and Beidou constellations to render as they have known conjunctions and significantly different orbital paths for variation in imagery. The sizes of RSOs drawn on to the rendered image are randomized between 3 to 5 pixels in size to best represent reference Sapphire data. Additionally, the brightness is randomized between magnitude 9 and 14.

2.2.3 BACKGROUND GENERATOR

The Background Generator creates the starry background of our simulated images. Leveraging the Tycho2 database, an astrometric reference catalog containing the positions of the 2.5 million brightest stars in the sky, we populated the background of each image with the exact stars that would fall into the FOV of the image satellite given its coordinates at the specified time. All the coordinates of the stars that fall within the view of the satellite payload are collected. Using the same process in place for the foreground satellite coordinates, the star coordinates are transformed from RA and Dec positions into pixel coordinates in the image. Once rendered, we tested the accuracy of our background images using plate solving algorithms, which were successfully able to determine the portion of the night sky rendered.

An interesting feature implemented in the background generator is the removal of background stars that would be drawn on an RSO. In real world scenarios, an RSO directly aligned with a star would be undetectable. In operational circumstances, we would likely analyze a series of multiple frames, allowing the RSO to move and become clearly visible once more. This feature eased the future training of the model, without compromising the probable operational flow.

2.2.4 VALIDATION

We utilized Systems Tool Kit (STK - AGI) to validate the position of all RSOs in our images. In STK, we built a simulation of all satellites propagated by our foreground generator, including Sapphire and all GEO satellites used in the images. We calculated Access Windows, times in which a satellite falls within Sapphire's FOV, and used these time stamps to generate the images. We also used STK's propagator to validate the accuracy of the Python propagator used in the simulator to calculate RSO coordinates.

2.2.5 NOISE

In order to simulate on-orbit imagery we added a variety of noise attributes. Initially, the image simulator generated RSOs and stars as two-dimensional circles with high contrast between the objects themselves and the dark space of the background. The resulting imagery could be likened to a black picture with white polka dots. As this is not representative of the source imagery, we compensated by adding a time dilation to simulate a realistic camera shutter speed and introduce noise that would be similar to conditions faced by a Low Earth Orbit (LEO) satellite. This addition ensured that the stars in the background appeared as streaks instead of symmetrical circles. The noise profiles added were multi-staged. The foreground has Gaussian noise added to 'soften' the edges of the RSOs. The background has Gaussian noise added to simulate Background Cosmic Radiation and to 'soften' the edges of the star streaks present. In addition, we introduced the option to add hot and dead pixels which is an unfortunate reality in image capture devices. Hot pixels are characterized by being always on, and dead pixels are always off. All this noise, time dilation and pixel options are configurable except for the noise added to the RSOs. RSOs will always appear to have a 'soft' edge regardless of the imaging device, but the remaining noise may be dampened by newer, more sophisticated noise dampening techniques and radiation hardening.

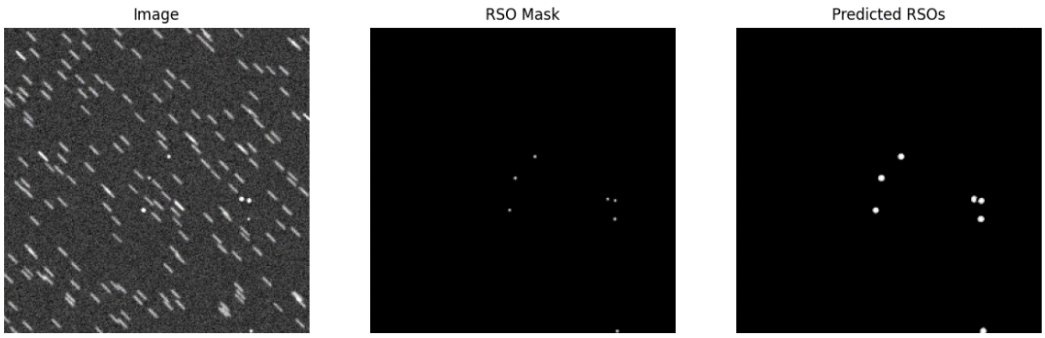

Figure 4-3 Sample Simulated Data. Left – Simulated Image. Middle – Truth Mask. Right – Predicted RSOs.

2.3 RSO DETECTOR MODEL

2.3.1 FRAMEWORK

The RSO Detector Convolutional Neural Network (CNN) was constructed with Tensorflow 2.6.2 and Keras 2.6.0 layers using Python 3.6. The basis of the model is from the *DeepTrack 2.0 U-Net Model* [1]. This model has proven to successfully identify small clusters of pixels in an image, the critical ability for the RSO Detector CNN. The RSO Detector CNN model has been optimized for the needs of the project. The U-Net model originally had approximately 31,000,000 trainable parameters and the RSO Detector CNN has 46,898, a reduction of 99.8%. This not only speeds up the training process, but also improves onboard performance of the model. Furthermore, it allows the training to devote more attention to the remaining weights. The convolution and convolution transpose layers are

used to determine the location of RSOs in the image. The model is compiled using the "Adam" optimizer, the main metric being "accuracy," and the loss function being the tensorflow_addons.losses.SigmoidFocalCrossEntropy().

2.3.2 LOSS FUNCTION

One major change to the original U-Net CNN, was the selection of the SigmoidFocalCrossEntropy loss function [1]. Focal loss functions are better suited for imbalanced datasets such as ours. Even with eight RSOs present in the image, it was seen that 99% or more of the pixels are not RSOs, but rather star streaks or background. This massive imbalance is partially compensated by the focal loss function as it considers this when training the model, leading to better weight values during training [2].

2.3.3 CNN MODEL TRAINING

The model was trained with simulated datasets. The final model saw 3484 images, 2800 of which were used in training, and the rest used in validation per epoch. A batch size of 32 was used during the run of 25 epochs. The model was then tested against other datasets. All images are 256x256 grayscale PNG files to save space and training time, without compromising on information captured in the image. Star streak length and direction, background noise, and quantity of RSOs differed between images to create a varied dataset. We found the aforementioned dataset size balanced the need for sufficient training data while avoiding overfitting. The selection of batch size and quantity of epochs was also a result of iterative investigations. Below 20 epochs, the model failed to predict consistently. More than 25 epochs showed at best marginal improvements, or led to overfitting.

2.3.3.1 OVERFITTING SIMULATED DATA

Overfitting and underfitting are both issues when it comes to these models, and in our case overfitting proved to occur more often. The sets of images used to train the model were developed so that each had a varying set of details that would provide enough unique or differing variables that the training would not begin to focus in on said details. Non-exhaustive examples of overfitting that were encountered during this project, and mitigated, are the following:

- Streaks always in one orientation, the same length and thickness
- All RSOs being a certain brightness, or always brighter than the stars
- A set amount of RSOs being present in the image, or the fact that an RSO is guaranteed to be present in an image

These issues were mitigated via introducing new training data with newly randomized details to prevent all the images from having similar features.

2.4 RSO DETECTOR RESULTS

2.4.1 DISTANCE METRIC

Traditionally, F1 and IoU scores are used to calculate a model's accuracy. Due to the imbalanced nature of our datasets, a method hinging on the quantity of object pixels vs. non-object pixels did not produce representative results of our prediction accuracy. Given the small size of RSOs in the simulated images, the distance between the center of the truth mask RSOs and the predicted RSOs was deemed to provide a more representative metric. The middle of each pixel cluster in either image was found via a centroid method and recorded in a list. The two lists were compared to see if an RSO had failed to be predicted, or if the prediction had false positives present. If the centers of these pixel clusters predictions were very close to a known RSO, the prediction was deemed a success. Should the centers not be close, we considered the model to have missed the target. In the images, the RSOs are usually 3 pixels in width, so the prediction was considered successful if the centers were within 5 pixels of each other. We found the centers of the prediction and the truth mask to be on average half a pixel off, demonstrating that the location of the predictions are extremely accurate. The distance metric demonstrates a more meaningful metric to measure the model's accuracy, as it is not based on the overall percentage of object pixels detected, but rather the quantity of RSOs in the image.

2.4.2 ACCURACY OF THE MODEL

We identified and used the most performant of our generated CNN models for subsequent evaluation of accuracy and error.

An analysis was conducted on the model's predicative certainty. The model ingests an image, normalizes the data from zero to one, and outputs a prediction normalized in the same manner. This threshold represents the model's certainty that the pixel in question is an RSO. Various thresholds were tested to identify which provided the best threshold, starting at pixels above 0.50 and going as low as 0.10. Figure 4-4 shows an analysis of the threshold's effect on the RSO Detector CNN's accuracy over multiple datasets containing star streaks of random lengths and widths, low to medium noise, and some with filtered edges. As the threshold increased, false positives trended to 0% occurrence but missed RSOs began to increase. When the threshold decreased, missed RSOs were reduced toward zero, however, false positives were reported more frequently. A threshold of 0.25 demonstrated the optimal balance between false positives and missed detections, which aligned with the overall maxima of successful predictions. This threshold was calculated specifically for the model selected for analysis. Should this experiment be repeated using a different model or on different data, this threshold would be tuned again through similar analysis.

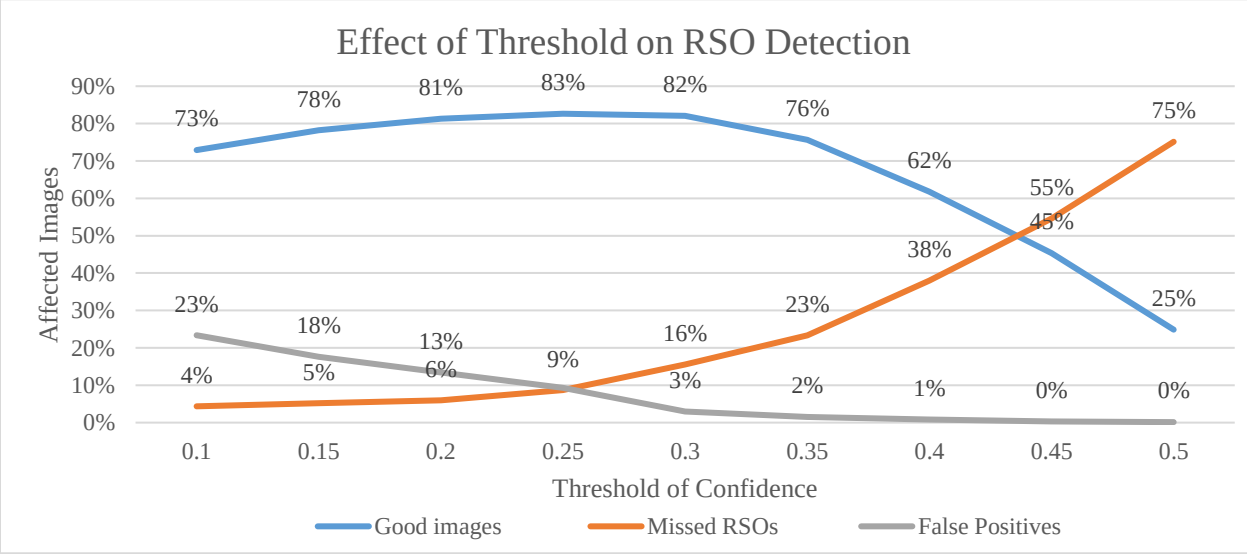

Figure 4-4 Effect of Threshold on RSO Detection

Using a CNN detector in an operational systems context both false positives and missed RSOs can be mitigated through the use of tracklet (or time series) data as well as complementary on-board data sources. If the given sensor was a first-order input the model, the model's threshold would be tuned to not miss RSOs and would use validation sub-routines for filtering false positives. For this research we opted to use a threshold that the intersection of the lowest number of both missed RSOs and false positives.

Several metrics were used to quantify the model's results. The "mean distance" refers to the amount of pixels between the center of the truth mask RSO and the predicted RSO. It speaks to the accuracy of the location positioning of the model's prediction. The metrics described in Table 4-1 outline the results seen from a validation dataset containing dimmer RSOs and limited background noise and no pronounced star streaking (sometimes found in imagery). The results from validation on a clean dataset are very good.

Table 4-1 Performance Results, First Dataset

Amount of images without issues	93.4%
Final mean distance	0.497 pixels
Amount of images with detected issues	6.60%
Average number of bad RSOs per bad image	1.13
Amount of images with a missed RSO	5.71%
Amount of images with a false positive	1.15%

We wanted to push the model further in order to understand its strength's and limitations. This was achieved by validation on datasets with more features and noise added than used in training. As one might expect we saw a slight reduction in accuracy, however, it still is accurate in its predictions. Table 4-2 shows the results from a dataset with much thicker star streaks, and much higher levels of background noise that the set used in training.

Table 4-2 Performance Results, Second Dataset

Amount of images without issues	83.8%
Final mean distance	0.527 pixels
Amount of images with detected issues	16.2%
Average number of bad RSOs per bad image	1.14
Amount of images with a missed RSO	11.8%
Amount of images with a false positive	4.78%

From this, the RSO Detector CNN has demonstrated an accuracy rate of 83.8%, where each prediction has a mean distance of 0.5 pixels from the truth mask center.

The detection error is driven by a multitude of factors. Sometimes the RSO was very dim, and the model failed to find it. In other cases, the RSO was close to a star streak and may have been misinterpreted as part of the star structure. Most false positive detections appear to occur at end points of the star streaks. It is possible the program misinterpreted the star streak's rounded edges as an RSO. Overall, errors appear with low occurrence even on noisy datasets, and it would be possible that further training with additional datasets or using time series data would improve the accuracy.

The mean distance between the predicted location and the truth mask location is extremely small, usually the prediction is only half a pixel off from the truth mask location. This means that the RA and Dec coordinates generated from the prediction are quite accurate and could be used in orbit calculations and propagations. With additional datasets for training and testing we could further decrease the error margin in the coordinates returned from the model's predictions.

2.4.3 POTENTIAL DRIVERS TO IMPROVE PERFORMANCE

To affect realistic model training and facilitate eventual feature additions and deployment, it will be necessary to obtain source imagery. The predictions on simulated imagery demonstrate a high degree of accuracy, recall and precision. Our simulated data and subsequent prediction accuracy allows us to validate the approach using anticipated operational input sources. When given 10 on-orbit Sapphire images, the accuracy of the RSO Detector CNN dropped to ~50%. Although our simulated images appear very realistic, a ML model is not performant to a type of dataset it has never seen. Access to broader temporal payload imagery in raw format would allow us to re-train and tune parameters, enabling significant maturation of the project. Given the success demonstrated thus far, we anticipate similar results for a CNN model trained on raw on-orbit payload data.

2.5 ONBOARD PROCESSING

The RSO Detector CNN was run on a Xavier Jetson AGX GPU [4]. A model file was created and its variables frozen. Subsequently, it was converted to the ONNX format for running inference on a GPU. This newly generated ONNX file was loaded onto the board and an executable was set up to ingest an image and use the model to run a prediction.

Once running, we experimented with the effects of the various power modes offered by the Xavier Jetson AGX. It sports various power budgets, CPU/GPU frequencies, amount of cores, and other options. There are 8 modes in total, as outlined in Table 4-3.

Table 4-3 Nvidia Xavier Jetson AGX Power Modes

Mode ID	0	1	2	3	4	5	6	7
Power Budget	Uncapped	10W	15W	30W	30W	30W	30W	15W
Online CPU	8	2	4	8	6	4	2	4
CPU Max Frequency (MHz)	2265.6	1200	1200	1200	1450	1780	2100	2188
GPU Max Frequency (MHz)	1377	520	670	900	900	900	900	670
DLA Max Frequency (MHz)	1395.2	550	750	1050	1050	1050	1050	115.2
PVA Cores	2	0	1	1	1	1	1	1
PVA Max Frequency (MHz)	1088	0	550	760	760	760	760	115.2
Max Frequency	2133	1066	1333	1600	1600	1600	1600	1333

Each of these modes was tested, and the model ran inference on a set of 100 test images. Inference speed refers to the time required for the CNN to process the image and make its prediction. The results outlined in Figure 4-5 are the findings based on the statistics of the varying power modes. Using floating point 32 data input in the reference, Mode 0 and mode 6 demonstrated the best results.

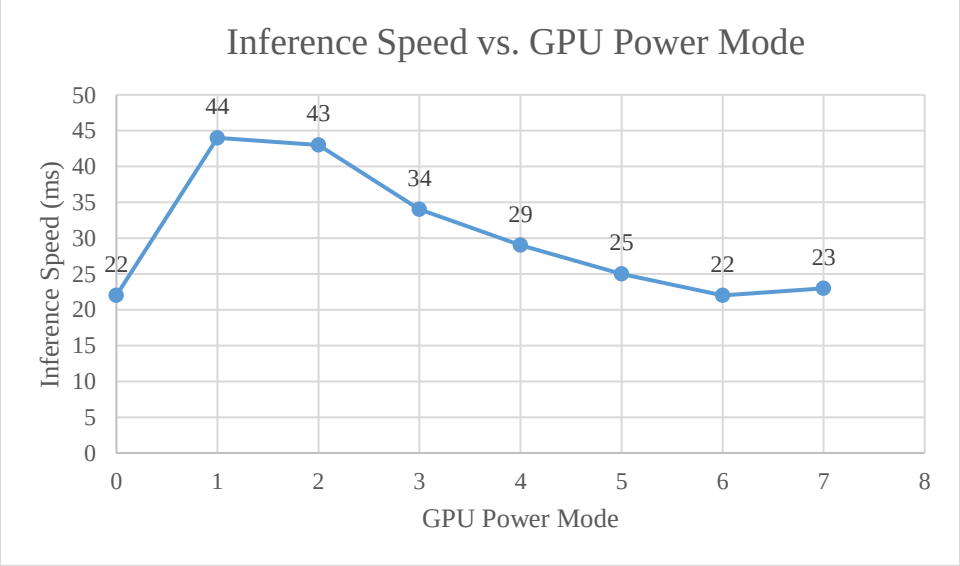

Figure 4-5 Inference Speed of the Power Modes

From the general trend depicted above, it appears the CPU frequency plays a higher role in the inference speed of the model rather than power allocated to the board. Power mode 7 most closely resembles similar space qualified board, and by extension, we should be able to make predictions in approximately 23ms. [5].

Figure 4-6 shows what one might expect, as the frequency of the CPU increases, the time required for inference decreases. It also shows that at lower frequencies, FP32 is faster than FP16, however as the CPU frequency increases, the difference becomes immaterial. Please note that the multiple datapoints at 1200MHz corresponds to the different power modes running at said frequency, as defined above in Table 4-3.

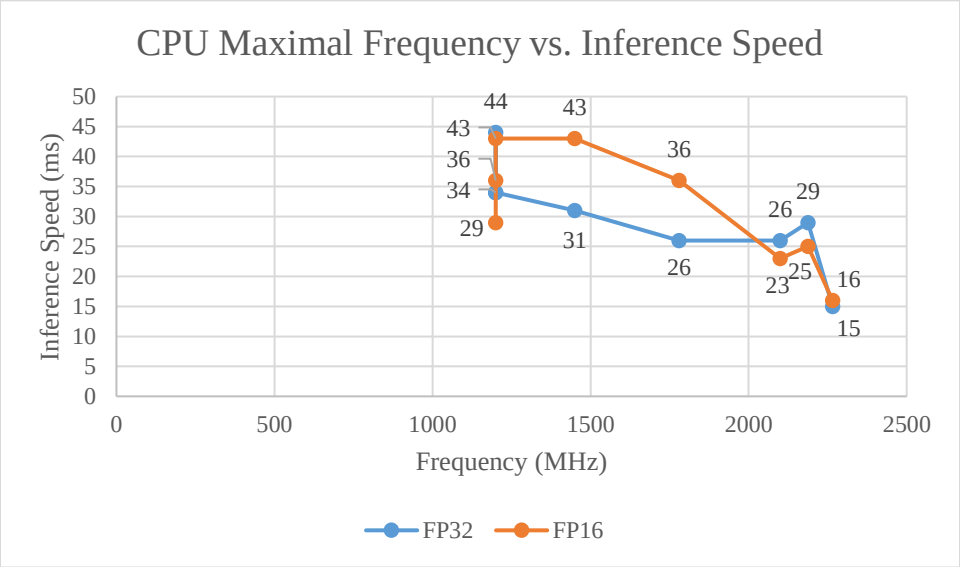

Figure 4-6 CPU Maximal Frequency compared against Inference Speed for FP32/16 Data

Once a final space qualified GPU is selected for flight, these analyses would be rerun to optimize for the exact hardware identified for the mission system.

3. RSO DETECTOR EXTENSIONS

The eventual progression of this project is to demonstrate a closed-loop decision support solution for on-orbit situational awareness and autonomous flight control.

In the near term, MDA is exploring future work associated with enabling the image simulator to model time series data including a broader range of optical payloads. MDA is also interested in the use of on-orbit SSA payload mission data comprised of broad temporal archives of image orientation, features and errors. This, in conjunction with images using wide FOV sensors, would provide a significant expansion of capability to the software. The datasets used in this prototype are modelled after images from Sapphire with a limited FOV. Training detection algorithms on images from hemispherical lenses would allow for the presence of many more objects in the frame and more computing workload. An example of a wide FOV sensor is MDA's hemispherical sensor HemiCam; this prototype is also developed in-house with support from a CSA STDP.

Currently we can adjust brightness configurations in the simulator to emulate RSOs with varying lighting conditions and incident angles. More configurable types of noise could also be added, such as simulating non-responsive pixels, whose output will fluctuate and not give consistent readings. Having broad temporal datasets would allow training of a CNN model using a full catalogue of changing visual magnitudes in RSOs as well as a greater collection of varying image features that could increase accuracy. The CNN's abilities could perhaps then extend to detect and track LEO satellites. Simulating these types of images is challenging due to the high relative speeds, poor light conditions and Earth's horizon, but would be interesting to test using sample on-orbit data.

Additionally, the ability for the software to use batch or tracklet images (images acquired over a number of seconds) would be more representative of operational payload data for SSA mission such as Sapphire. These image segments will improve detection accuracy where RSOs may overlap with a background star for a single image frame. The preliminary results were adjusted to take this into account as it is assumed that time-series input data would mitigate this challenge.

Another key objective relates to the anticipated on-orbit hardware itself. To date, we have been running our algorithms on the Nvidia Xavier Jetson AGX, a powerful GPU, but not one qualified for space flight. We have researched available options on the market and the Innoflight CFC-500 in an example of a suitable candidate. Similarly to the Xavier Jetson AGX, the CFC-500 also supports the Compute Unified Device Architecture (CUDA) runtime environment. This environment should allow our algorithms to transfer with minimal complications. The low power requirements of this chip will be an asset given the satellite's constrained system. It also sports two Nvidia Tegra K1 chips, which should decrease our inference times and handle a broader system.

4. FUTURE WORK

While RSO detection from solely optical images is extremely valuable, supplementary capabilities will be necessary to implement an entire on-orbit SSA system. We can discern RA and Dec coordinates from optical images, but range information remains crucial for orbit determination, and by extension, collision risk assessment. The introduction of multi-modal input types such as radar, LiDAR, or even stereoscopic images would be an asset to an SSA system. With multi-modal inputs, we can create a system that would enable a spacecraft to create its own situational awareness crucial to closed-loop decision support systems.

5. CONCLUSIONS

The SSA OBP project has developed and demonstrated an end-to-end prototype system for the onboard detection of RSOs in optical image data. It has been developed using a ML algorithm created with simulated images of resident space objects amongst a background of star streaks in space. Utilizing thousands of simulated images, we successfully trained the modified U-Net CNN with a 99.8% reduction of weights from the standard model. Varying details of the simulated images allowed us to show the model can successfully predict on a multitude of lighting conditions. Accuracy of the model was assessed by measuring the distance between the center of the predicted RSO and the center of the truth mask RSO. Verification of the predicted images showed that the location of predicted RSOs are extremely accurate, often not more than half a pixel off-centered, with an 83% accuracy. When this model was run on a GPU, it could run inference at about a speed of 23ms at around 15W power budget and 2000 MHz maximal frequency. The model is able to be retrained for different input image sources. This project has laid the groundwork for utility in the SSA domain, from an initial collision detection towards autonomous maneuvering decisions without regret.

6. REFERENCES

The following articles reviewed on this project are listed.

[1] Softmatterlab, "DeepTrack-2.0/examples/GET-started at develop softmatterlab/deeptrack-2.0," GitHub. [Online]. Available: https://github.com/softmatterlab/DeepTrack-2.0/tree/develop/examples/get-started. [Accessed: 20-May-2022].

[2] "Tfa.losses.sigmoidfocalcrossentropy: tensorflow addons," TensorFlow. [Online]. Available: https://www.tensorflow.org/addons/api_docs/python/tfa/losses/SigmoidFocalCrossEntropy. [Accessed: 20-May-2022].

[3] E. Tiu, "Metrics to evaluate your semantic segmentation model," Medium, 03-Oct-2020. [Online]. Available: https://towardsdatascience.com/metrics-to-evaluate-your-semantic-segmentation-model-6bcb99639aa2. [Accessed: 20-May-2022].

[4] "Jetson AGX Xavier Developer kit," *NVIDIA Developer*, 25-Jan-2022. [Online]. Available: https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit. [Accessed: 20-May-2022].

[5] *CFC-500*. [Online]. Available: = https://www.innoflight.com/product-overview/cfcs/cfc-500/. [Accessed: 20-May-2022].