Challenges and Opportunities for CubeSat Detection for Space Situational Awareness using a CNN

Jim Aarestad, Andrew Cochrane, Matthew Hannon, Evan Kain, Craig Kief, Steven Lindsley, Brian Zufelt
COSMIAC
University of New Mexico
Albuquerque, NM; 1+ (844) 267-6422
steve.lindsley@cosmiac.org

ABSTRACT

The deployment of artificial neural networks (ANNs) on small satellites will improve space situational awareness (SSA) where scarce radio resources limits the interactions between space-born and ground-based systems. The application of ANNs in space is stymied by lack of curated datasets. This project addresses an ongoing problem of small spacecraft detection and identification. This paper will present the problem and the activities the team has taken to advance this field.

INTRODUCTION

CubeSats are a class of nanosatellites that began to appear in the space community in June of 2003. Multiple organizations have announced plans to launch hundreds of these nanosatellites (most of which weigh less than ten kilograms) into low Earth orbit (LEO) in the next few years. Government space agencies engage in numerous earth observation activities that require a mechanism to be able to identify objects and categorize them. [11,12] Past activities required the download of many images across limited radio links so that ground-based systems could perform classification. This solution occupies already scarce bandwidth and increases the power usage of the spacecraft from increased communication. Artificial neural networks (ANNs) now make possible the use of on-orbit resources for this process. The usage of ANNs also introduces the need for significant, power-intensive computational resources for model training, testing, and validation. However, a fully trained model has already undergone these steps and only needs to perform inference during regular operation. The inference is significantly less computationally intensive than training because only a single forward pass through the network is needed. Therefore, a fully trained model can be deployed and utilized in orbit on systems featuring only small, power-efficient processing platforms designed for CubeSats.

A significant component in the process of training an ANN is that of obtaining a dataset. While there are many freely available datasets, no such dataset exists for CubeSat images. This paper discusses the process of creating a CubeSat image dataset, including the several methods used to collect images and the tools that we used to organize and label the observations. A brief discussion of the myriad of tools available to create and train ANNs concludes in a workflow that is capable of both training and deploying the ANN.

BRIEF BACKGROUND OF CUBESATS

A CubeSat is a small satellite or spacecraft based on a 10 cm × 10 cm × 10 cm size per unit (U). CubeSats are configured in multiples of units to create a size range from 1U to 12U, with 1U, 3U, and 6U being the most commonly used configurations (Figure 1).

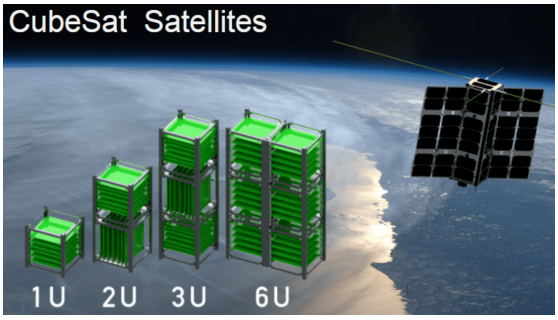

Figure 1. CubeSat Configurations

(image credit: http://www.spaceschool.ca/satellites.html)

This concept of standardized size came after the success of the OPAL (Orbiting Picosatellite Automatic Launcher) mission, which consisted of a primary satellite which, once deployed by the lift vehicle and stationed on orbit, subsequently deployed six small picosatellites. The picosats ranged from 7.5 × 10 × 2.5cm (short) to 7.5 × 20 × 2.5cm (long) with two picosatellites in the long configuration and four picosatellites in the short configuration. The picosatellites were then independently tested to prove the capability in utilizing smaller satellites as secondary studies or missions. This endeavor was executed by Stanford University, which designed the modern standard for CubeSats and California Polytechnic State University (Cal Poly), which designed a launcher to deploy them into orbit.

At the time CubeSats were first introduced, traditional satellites were large and prohibitively expensive to develop and launch, creating an effective, insurmountable barrier to space-based research for most organizations. These new small satellites were designed with the objective of creating an affordable platform that would be more readily accessible to research and education efforts. In addition to affordability, the CubeSat designs offered a structure (or *bus*), basic electrical power, control, and communications functionality to allow users to focus more on the development of the payload and less on the spacecraft [1]. Within this framework, designers are able to rapidly test new ideas and technologies that might otherwise be too costly or risky for conventional satellites. CubeSats can be deployed using a variety of methods, including small rockets, secondary payloads aboard larger missions, a Poly-Picosatellite Orbital Deployer (P-POD) onboard the International Space Station, or even by an astronaut during extra-vehicular activity.

Organizations ranging from amateur radio clubs to universities and commercial entities have launched CubeSats. NASA's CSLI (CubeSat Launch initiative) has been set up to promote STEM education that launches CubeSats as auxiliary payloads on already scheduled missions. CubeSats have found a place in deep space missions such as the Mars Insight probe that used two 6U satellites for communications relay. The number of launches of small satellites has grown exponentially due to the rising popularity of CubeSats. As of this writing, more than 1,000 have been deployed, the majority of which remain in orbit. More than half of these CubeSats will be in orbit for five years or longer, and some will remain in orbit for twenty years. Future missions will place CubeSats in orbit around the Earth's moon and send them deeper into our solar system.

The burgeoning number of CubeSats on-orbit represents a potential hazard to future space missions. There are currently almost 900 CubeSats in orbit [8], and with over 70 nations along with numerous academic and commercial sector entities launching space vehicles, this number is expected to grow significantly in the coming years [9]. Therefore, the ability to detect, classify, and track CubeSats will become essential in order to protect space-borne assets. Traditionally, ground-based systems with heavy computational capability or humans-in-the-loop are used for image-based identification of space objects. Often, the imagery of CubeSats is captured as they are deployed, but these approaches are not feasible for automated detection of CubeSats, a looming necessity, especially if manned space missions are to resume.

Ground-based space object classification systems have been developed to use ANNs that ingest imaging or radar data [1] to classify objects interacting with the atmosphere. Physics-based models have been implemented that use data from laser range finders [2]. Some work has been done to implement machine learning systems aboard small satellites for onboard data reduction [3], demonstrating an associated cost reduction. A simulation of a space-borne algorithm that used a camera to detect entry events [4] demonstrated successful detection using space-based processing capabilities. These experiments lend credence to the effort of implementing ANNs for space-resident object detection and classification.

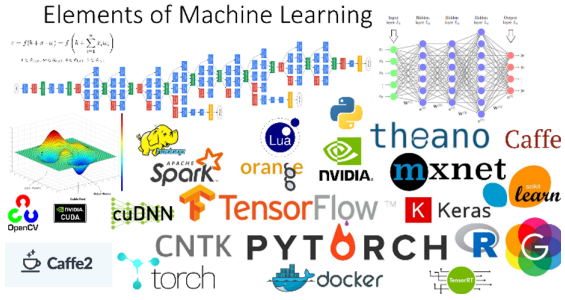

Figure 2: Elements of Machine Learning Development and Deployment

MACHINE LEARNING TOOLCHAIN, TRAINING, AND DEPLOYMENT

Simply possessing a working knowledge of the math and methods for the application of an artificial neural network is generally insufficient to develop deep learning methods for new application domains. For years, a gap has existed between those who have conducted the research to develop new models and

those who would deploy those models on custom-built hardware. A project that developed an ANN would often develop custom, bare-metal code that was difficult to reuse. Even now, the community is fragmented; Figure 2 shows a collage of the many different tools, frameworks and interfaces available for development, data conditioning and model training. Before training the model, an engineer must select from these tools to develop a toolchain for development and a pipeline for training that is capable of conditioning data and deploying the trained model to a small, power-efficient computation platform. [23] Figure 3 provides a flowchart of the pipeline. The pipeline developed for this project is described below.

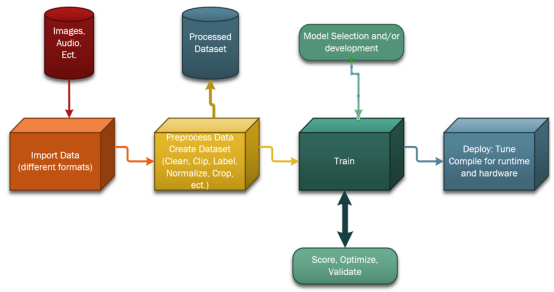

Figure 3: Flowchart Outlining the Pipeline for any Deep Learning Project

Data Collection

In order to train an ANN for object detection, large amounts of data are required, this training process requires hundreds or even thousands of images to construct an accurate classification model. For this model to work correctly in most situations, the dataset needed to contain images with many different CubeSat form factors, designs, and components. Given the task of collecting as many images as possible, two methods of image acquisition were employed: 1) image collection by sifting the internet for CubeSat images, and 2) generating synthetic images from computer-aid design (CAD) models and rendering computer-generated images (CGI).

The image collection process involved searching through COSMIAC's repository of past CubeSat projects and various online resources that contained CubeSat images. One technique that was particularly effective for improving search results was to change the default search language to common European and Asian languages. Once located, online repositories that contained CubeSat images were scraped using HTTrack Website Copier (https://www.httrack.com/). Sites such as EOPortal (https://www.eoportal.org/) contain many images that were useful to train the ANN to detect CubeSats in their development and testing environments, which helps to prevent overfitting which can occur if images of only one background were used. One of the best resources for CubeSat images taken during deployment in space was NASA's CubeSat repository, an example of which is shown in Figure 4. Scripts were also developed to download images using search engines such as Google Images. Data gathered this way did not contain the label metadata necessary for classification (1U, 2U, etc.), and some of these were used to validate the trained ANN.

Figure 4: CubeSat above Earth

A combination of CGI tools such as OnShape, Blender, Substance Painter, and Unreal Engine 4 enabled the team to generate synthetic images to increase the amount of and augment the collected images for a more generalized dataset. 3D models were acquired from both the COSMIAC repository figure 5 and NASA 3D Resources (https://nasa3d.arc.nasa.gov/models) as well as generated using OnShape CAD software and component models from Pumpkin Space Systems (https://www.pumpkinspace.com). Some models from NASA, such as MiRaTA, required conversion into a usable format, stereolithography (.stl), that could be imported into Blender. Once imported, a UV layer could be baked, and the whole model exported as an

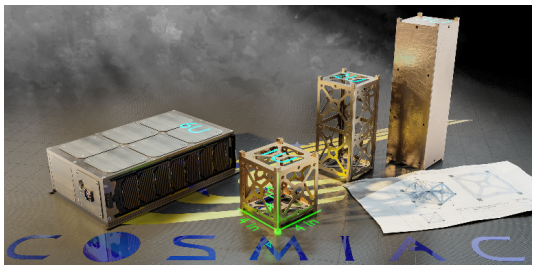

Figure 5: Computer Generated Image of Multiple CubeSats

object file (.obj) to be used in Substance Painter for applying a final texture to the model. While Blender provides node based texturing, the team found that Substance is an easier workflow that provided faster results. After texturing, Blender was used to generate a movie where viewing angle, distance from the object, focal length, and lighting was constantly changing to produce a video under different conditions. With some slight editing each frame of the video was used as a single image, which quickly expanded our dataset. This was done in the cycles rendering engine that would allow for a transparent background behind the CubeSat model seen in figure 6. The transparent background would allow for two key benefits when preparing the images for the dataset: 1) automatic placement of the box bounding for annotation used during training of the ANN, and, 2) control of the background noise to reduce overfitting in the model.

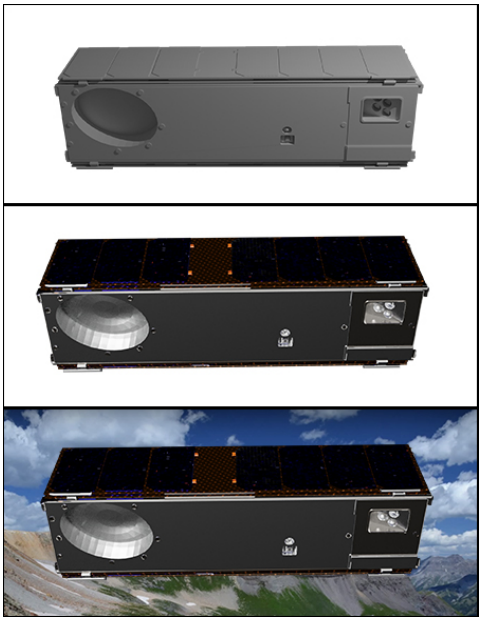

Figure 6: NASA MiRaTa Model: Imported, Textured, Noise Added

A script was created to look at the values within the alpha layer of each generated image to find the pixels that contained just the CubeSat within the frame. These pixel location values were then used to create a text file that specified the bounding box used by the model to identify the object of interest during the training. This saved the team a large amount of time by automating the bounding box process; by comparison, the data mined pictures had to be boxed by hand for every single image.

Additionally, with the transparent background, the noise could be added, and the type of noise could be changed for the synthetic images. A dataset using only a single type of background, such as a pure color like black or red, will produce overfitting, and the model will start to require that color in the background to detect the CubeSat accurately. In solving this problem, the rendered CubeSat movie was used as the top layer with a series of random video clips from the Internet playing in the background. This allowed for quick creation of random noise to prevent the background from becoming a part of the trained requirements for positive detection. After the random noise had been added to the synthetic CubeSat images in a final video, each frame was then split into a portable graphics format (.png) image and reunited with its proper metadata file containing the bounding box data.

Once a collection of CubeSat images was acquired, some manual processing was necessary to ensure the images were in the proper format to be ingested by a neural network training framework. Our team performed the laborious task of creating the image metadata for the images collected from the Internet. Image processing software was used to scale, shift, and rotate existing CubeSat imagery to augment our training dataset.

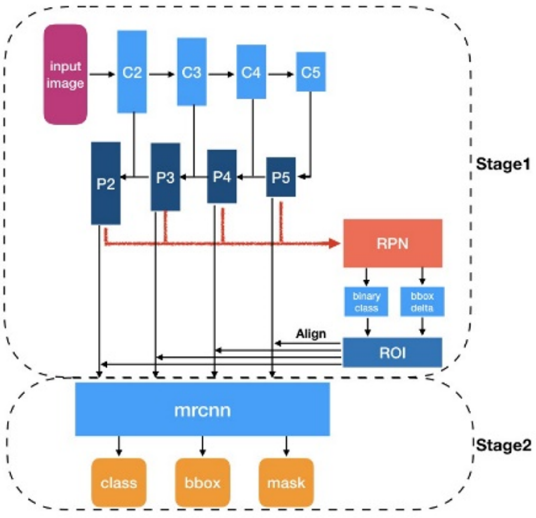

Figure 7: Mask RCNN Model Structure

A dataset extensions strategy was employed whereby the existing images were corrupted by additive and multiplicative noise of Gaussian and Laplacian distributions, respectively. These noisy images were further processed with regular scaling, shifting, and rotating operations to yield an extended training dataset.

Model Selection, Training and Validation

Two standard object detection and classification models, Mask RCNN [5] and YOLOv3, were chosen for their popularity and demonstrated capability for the detection of diverse types of objects. Mask RCNN (Regional Convolution Neural Network) is an object detection model that uses a two-stage process whose structure is illustrated in Figure *7*. The incoming image or video frame is examined across multiple regions to generate possible areas where an object may be placed. Stage One uses a Feature Pyramid Network [7], which allows the image to be examined multiple times. It should also be mentioned that Mask RCNN draws upon common feature extractors such as VGG and ResNet. These feature extractors comprise the backbone of the model. The second stage checks these regions for objects and produces a bounding box over found objects. It should further be noted that Mask RCNN is not a real-time model since multiple examinations of the image are required. For this reason, Mask RCNN is used to process stored information and cannot act upon real-time streaming data.

For real-time detection, a model known as YOLOv3 (You Only Look Once Version 3) was chosen. The model structure is shown in Figure 8. Unlike Mask RCNN, YOLOv3 provides a single examination of the incoming data and splits the inference into three outputs. The outputs constitute the detection of objects at different scales. These scales are generated by splitting the image into grids of different sizes.

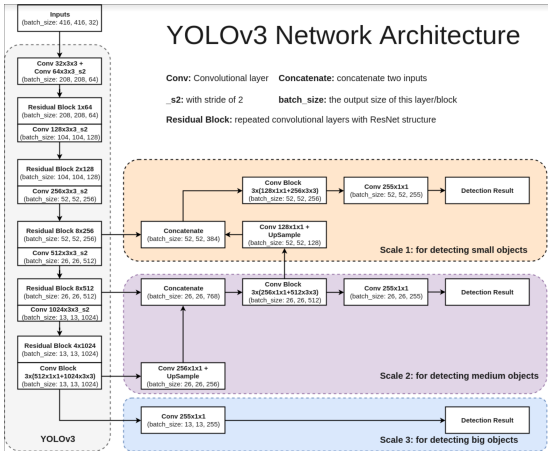

Figure 8: YOLOv3 Model Structure

Both models have been proven to perform well on many different platforms, and their performance has been documented in several papers. For this project, the objective was to create a pipeline that offered the quickest method to move from concept to deployment. With the models selected, a framework was chosen to implement and train the models.

There are many frameworks available (TensorFlow1, TensorFlow2, Caffe, Theano, Torch, MXNet, etc.). In addition, there are many toolsets that come with applications such as MATLAB and R. However, this project required a toolchain which covered the entire workflow, including development, training, testing, and model deployment. Some tools, such as MATLAB, are capable of model development, training, and testing, but do not provide an easy means of porting the trained model to an embedded computing environment.

The team chose to use TensorFlow1 with Keras for high-level model development. Keras provides a high-level abstraction of the TensorFlow framework used to develop and train the network as well as examine model performance, while TensorFlow provides a framework to efficiently map tensor operations to available computing resources in the target hardware.

The process of training a machine learning (ML) model involves providing a machine learning algorithm (the learning algorithm) with training data to learn from. The team built a large CubeSat library for this purpose as one didn't already exist. The term ML model refers to the model artifact that is created by the CubeSat training process.

The training data contained the correct answer, which is known as a target or target attribute (1U, 2U, 3U, or 6U). The learning algorithm finds patterns in the training data that map the input data attributes to the target (picking a 6U out of the noise), and it outputs an ML model that captures these patterns. You can use the ML model to get predictions on new CubeSats for which you do not know the target.

Model validation is the process of evaluating a trained model on a test data set. There are many different types of validation options that are understood and repeatable. The team chose to use the K-fold model for validation purposes. K-fold uses a method called cross-validation. Cross-validation is a statistical method used to estimate the skill of various machine learning models like the one developed during this activity. Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. K-fold has a single parameter called "k" that refers to the number of groups that a given data sample can be split into. The procedure is often called k-fold cross-validation. When a specific value for k is chosen, it may be used in place of k in reference to the model, such as k=10 becoming 10-fold cross-validation. The team used K-fold to estimate the skill of our machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in

general when used to make predictions on data not used during the training of the model. It is a popular method because it is simple to understand and because it generally results in a less biased or less optimistic estimate of the model skill than other methods, such as a simple train/test split. The team was pleased with the results of our validation, but when we presented the work to our government contacts, they said it would be necessary to increase the sample size to a minimum of 50,000 samples. While we agree a large number of samples are required, the application (video, audio, radio, image, text, etc.) must be taken into consideration, in addition to the way the samples were collected or generated. A dataset consisting of 50,000 images is great, where 50,000 samples of raw quadrature signal (often called IQ) radio frequency data is not enough. [13,14,15,16,20,21]

Deployment

There are many options for porting a trained model to compact hardware for use in the field. This is a process called deployment and involves: 1) the optimization of a data ingest engine which feeds sensor data to the model for inference, and 2) the optimization of the inference engine itself. Many runtime products are optimized to take advantage of available hardware resources for inference speed up. Three great examples of optimization products are TensorRT for hardware with an Nvidia GPU, Intel's OpenVINO platform that also works on Intel's embedded Neural Compute Stick (NCS2) platform, and Nvidia's Jetson AGX Xavier shown in figure 9. The Movidius™ Neural Compute Stick (NCS) is a tiny fanless deep learning device that you can use to deploy AI programming at the edge. The NCS2 enables rapid prototyping, validation, and deployment of DNN inference applications and could easily be transferred to a CubeSat or Nanosatellite for instant computing capability. [22] Xavier is designed for the next generation of autonomous machines and is perfectly suited for object detection and many other AI inferencing capabilities. With configurable operating power modes, the Xavier can be set to use 10W, 5W, or 30W, making it a perfect candidate for small satellites. [17,18,19]

For this project, we chose to use the OpenCV (Open Source Computer Vision Library) DNN engine. Deep Learning is the most popular and the fastest growing area in Computer Vision today. Since OpenCV 3.1, there has been a DNN module in the library that implements forward pass (inferencing) with deep networks, pre-trained using some popular deep learning frameworks, such as Caffe. OpenCV is an open-source library used to perform computer vision tasks. It offers over 2500 computer vision algorithms, including classic statistical algorithms and modern machine learning-based techniques, including neural networks. OpenCV boasts a community of almost 50,000 developers and over 18 million downloads. OpenCV is used by huge companies like Google, Yahoo, Microsoft and Intel, research bodies, governments, and also startups and individual users. While it used to be difficult to learn and use, usability and documentation are gradually improving, and the team doing this work was able to interface and begin using the tool quickly.

Figure 9: Left - Raspberry Pi with NCS2, Right-Nvidia Jetson AGX Xavier

While this is not the most optimized inference engine, it provides the greatest flexibility. OpenCV supports Deep Learning frameworks Caffe, Tensorflow, Torch/PyTorch. With OpenCV, you can perform face detection using a pre-trained deep learning face detector model, which is shipped with the library. Our goal was to use it for CubeSat detection.

OpenCV DNN 4.1.2 can run on ARM and x86 CPUs and will take advantage of GPU resources with OpenCL. One limitation with OpenCV DNN is that (as of the time of this paper), it cannot directly read Keras H5 file format, which stores the model's layer structure and weight matrices that are produced during training. However, given that OpenCV 4.1.2 is able to read TensorFlow1 files, and Keras uses theTensorFlow1 backend, a method exists for converting the data to the required format.

ADDITIONAL DETECTORS

The COSMIAC team has applied a similar data collection and neural network training process to identify satellite components. A second detector was trained to identify solar arrays on not only CubeSats but many different types of spacecraft in orbit. The CubeSat dataset was curated to exclude any images that did not contain any visible photovoltaic cells. The same process of data mining and synthetic imaging was applied to create a suitable dataset that included

traditional satellites and other space objects with solar power.

Ensemble learning, a process in which predictions from multiple neural network models are combined, can be applied not only to reduce variance and error but also to understand inputs from one or more senor(s). As an example, a CubeSat is detected by the first detector, and secondary ANN is then utilized to detect if the output clip within the bounding box of the original detector also contains photovoltaics. If both networks show positive results, the chances of a false positive are greatly reduced before restrictive and costly data transmissions are utilized.

Additionally, this will allow for more complex problems to be evaluated like a CubeSat's potential mission in orbit. A small CubeSat that is detected might be classified as 1U size and then verified as possessing solar cells with a small power need that could likely be used as a beacon for student research. A different CubeSat is classified as a 6U and is detected as possessing a large solar array, which would fall into a class of more powerful satellites like Asteria used for exo-planet study shown in figure 11. A deeper understanding of situational awareness can be inference, and estimates can be made about space domain awareness (SDA). SDA is the new term for the former programs called SSA.

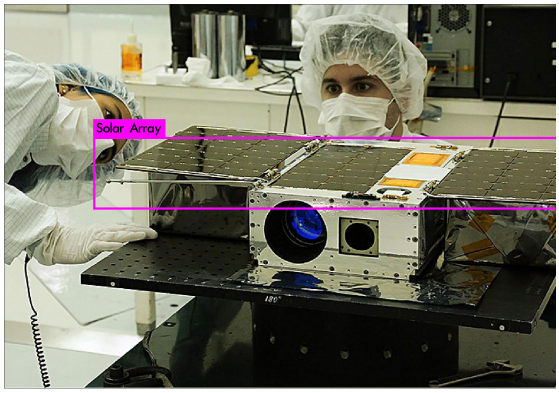

Figure 11: Trained Solar Array Detector – Asteria (image credit : NASA/JPL)

CONCLUSIONS

The available machine learning models, combined with off-the-shelf computing that is small yet, powerful, are ready to be utilized in a space environment. The deployment of artificial neural networks (ANN) for use in spacecraft classification applications is challenging due to the lack of curated datasets. Making of these libraries is possible but time-consuming to do correctly. The COSMIAC team has curated an initial dataset for the detection and classification of CubeSats ranging from 1U to 6U. Early training and validation have shown promising results on a variety of computing platforms. A plan for further developing the ANN training and deployment workflow has been presented. There remains significant work to be completed before this system can be considered mission-ready. The available machine learning models, combined with off-the-shelf computing that is small yet, powerful, are ready to be utilized in a space environment. Additionally, work will need to be developed on optical systems. Picking up a nanosatellite from across the room is one thing, but capturing something like this moving five miles a second is entirely different. It is possible but will require significant funding. In our discussions with the government, we were informed that in their opinions, everyone does machine learning for space awareness, but none of them do it well.

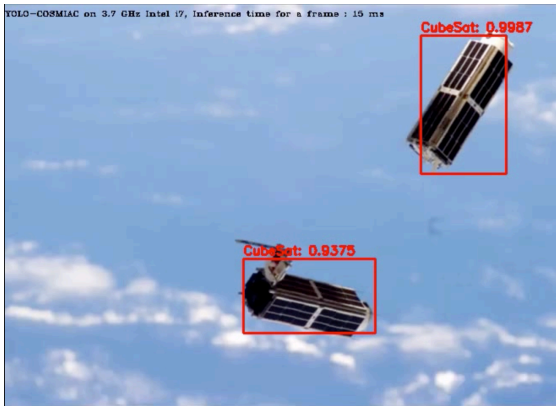

Figure 10: Trained CubeSat Detector YOLOv3 experiment

FUTURE WORK

The team will expand and improve the dataset to achieve better training to correct any gaps in inference and create a more robust detector. A classifier will be created to determine what size of CubeSat has been detected. The team will optimize the neural network for deployment by pruning and other technics to achieve a balance of speed and accuracy suitable to fly the developed system in orbit. This would be comprised of a self-contained box with an imager, a processor, and a downlink channel. One concept of operations would be to have the system detect a CubeSat as it passes by, capture an image, characterize it, and download the images to an earth-bound location. Most CubeSats have a deorbiting requirement of not more than 25 years beyond the end of its mission. Since "end of mission" can often be ill-defined, the number of nanosatellites in

orbit will necessarily continue to grow and present orbital debris hazards.

References

1. R. Linares and R. Furfaro, "Space Object Classification Using Deep Convolutional Neural Networks," in 2016 19th International Conference on Information Fusion, ed. Heidelberg, Germany: IEEE, 2016, pp. 1140-1146.
2. M. Nayak, J. Beck, B. Udrea, and Ieee, "Design of Relative Motion and Attitude Profiles for Three-Dimensional Resident Space Object Imaging with a Laser Rangefinder," in IEEE Aerospace Conference, Big Sky, MT, Mar 02-09 2013, NEW YORK: Ieee, in IEEE Aerospace Conference Proceedings, 2013.
3. P. Hershey, B. Wolpe, J. Klein, C. Dekeyrel, and Ieee, "System for Small Satellite Onboard Processing," in 11th Annual IEEE International Systems Conference (SysCon), Montreal, CANADA, Apr 24-27 2017, NEW YORK: Ieee, in Annual IEEE Systems Conference, 2017, pp. 23-28.
4. R. T. Nallapu et al., "Smart Camera System On-board a CubeSat for Space-based Object Reentry and Tracking," (in English), 2018 Ieee/Ion Position, Location and Navigation Symposium (Plans), Proceedings Paper pp. 1294-1301, 2018
5. K. He, G. Gkioxari, P. Dollar, and R. Girshick, "Mask R-CNN," arXiv, 2017.
6. J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," arXiv, 2018.
7. T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature Pyramid Networks for Object Detection," arXiv, 2016.
8. Harvey W., Garon S., Martens S., Brassard G., Tafazoli S., Souza M., Bédard D., Scott R.L., Wallace B. "The Near Earth Object Surveillance Satellite (NEOSSat)", Astro 2006, Montreal, Quebec, April 2006.
9. M. J. Holzinger and M. K. Jah, "Challenges and Potential in Space Domain Awareness," Journal of Guidance, Control, and Dynamics, vol. 41, no. 1, pp. 15-18, 1 January 2018. DOI: 10.2514/1.G003483
10. Wallace, B. et al. "The Near Earth Orbit Surveillance Satellite (NEOSSat). International Astronautical Congress 2004. Vancouver.
11. Shoemaker, Jim. "Space Situational Awareness and Mission Protection." Aug. 2005. 2008. http://www.darpa.mil/DARPAtech2005/presentations/space_act/shoemaker.pdf>
12. "Navy Space Surveillance System [NAVSPASUR]." GlobalSecurity.org April 2008. https://www.globalsecurity.org/space/systems/navspasur.htm
13. Girimonte, D and D. Izzo, "Artificial Intelligence for Space Applications," in Intelligent Computing Everywhere, Springer, London, 2007, pp. 235–253
14. Krizhevsky, A., Sutskever, I., and G. Hinton, "ImageNet classification with deep convolutional neural networks," Proceedings of the 25th International Conference on Neural Information Processing Systems, Vol. 1, Dec 2012, pp. 1097-1105.
15. Zoph, B., Vasudevan, V., Shlens, J., and Q. V. Le., "Learning transferable architectures for scalable image recognition," arXiv preprint arXiv:1707. 07012, 2017.
16. Rumelhart, D. E., Hinton, G. E., and R. J. Williams, "Learning representations by back-propagating errors," Nature International Journal of Science, Vol. 323, No. 6088, Oct 1986, pp. 533–536.
17. Schartel, A., "Increasing Spacecraft Autonomy through Embedded Neural Networks for Semantic Image Analysis," M.S. thesis, Luleå University of Technology, Luleå, Sweden, 2017.
18. Buonaiuto, N., Kief, C., Louie, M., Aarestad, J., Zufelt, B., Mital, R., Mateik, D., Sivilli, R., and A. Bhopale, "Satellite Identification Imaging for Small Satellites Using NVIDIA," Proceedings of the AIAA/USU Conference on Small Satellites, Launch, SSC17-WK-56.
19. Grabowski, J., "Neuromorphic Computing for Classification in Optical and SAR Data with IBM TrueNorth and Xilinx FPGAs," 10th Workshop on Fault-Tolerant Spaceborne Computing Employing New Technologies, Albuquerque, NM, May 30 – Jun 2, 2017.
20. Mackinnon, J., Ames, T., Mandl, D., Ichoku, C., Ellison, L., Manning, J., and B. Sosis, "Classification of Wildfires from MODIS Data using Neural Networks," Machine Learning Workshop, Aug 2017.
21. Farmer, M., Parker, G., King, L.B., Radecki, P., and Katalenich, J., "Nanosatellite Attitude Control System for the Oculus: A Space-Based Imaging Platform for Space Situational

Awareness" SmallSat Conference, 2008, SSC08-XII-8.

22. Duijn, P., Redi, S., "Big Data in space" Conference on Small Satellites, SSC17-I-08

23. Johnson, S., Corcoron, M., Rohrer, M., "Platforms Designed for Big Data Provisioning with Small Satellite Constellations" Conference on Small Satellites, SSC17-XI-07