SSC22-WKIV-03

Machine Learning Image Processing Algorithms Onboard OPS-SAT

Shreeyam Kacker, Alex Meredith, Kerri Cahoy
Massachusetts Institute of Technology
77 Massachusetts Avenue, Cambridge MA 02139; +1-617-253-7805
shreeyam@mit.edu

Georges Labrèche
Tanagra Space
Queens, NY; +1-919-912-9136
georges@tanagraspace.com

ABSTRACT

We discuss the deployment of image processing algorithms developed for BeaverCube-2, a project under development between the MIT Space Telecommunications, Astronomy, Radiation (STAR) Lab and the Northrop Grumman Corporation. The algorithms were uploaded to and executed on OPS-SAT, a 3U CubeSat owned and operated by ESA with a processing payload that allows rapid prototyping, testing, and validation of software and firmware experiments in space at no cost to the experimenter. Testing these algorithms onboard OPS-SAT significantly reduces risk for future on-orbit image processing missions such as BeaverCube-2. We focus on four image processing algorithms used for cloud detection: a luminosity-thresholding method, a random forest method, an U-Net based deep learning method — all developed by STAR Lab for BeaverCube-2 — and a k-means clustering deep learning method implemented by the OPS-SAT Flight Control Team (FCT). We evaluate each method in terms of in terms of overall accuracy, power draw, and temperature rise on-orbit, and discuss the challenges of implementing these methods on embedded hardware and the lessons learned for BeaverCube-2.

INTRODUCTION

Improvements using machine learning for image processing have the potential to enable more autonomy for applications where communications latency is a challenge. Identifying and discarding undesirable images on-orbit allows satellite operators to downlink only high quality images of a target region, saving time and resources such as power, bandwidth, and operator effort. In this work, we explore the problem of on-orbit cloud segmentation with limited computational resources on the BeaverCube-2 and OPS-SAT missions.

BeaverCube-2 is a mission jointly developed by the MIT Space Telecommunications, Astronomy, and Radiation (STAR) Lab and the Northrop Grumman Corporation which aims to demonstrate the use of an Artificial Intelligence (AI) Computational Accelerator System-on-a-Chip (SoC) on a 3U CubeSat in Low-Earth Orbit (LEO). BeaverCube-2 will leverage this AI accelerator to perform on-orbit image processing to identify clouds and ocean fronts around the Cape Hatteras region of North Carolina.[1,2] The concept of operations for the mission is shown in Figure 1. Previous work has been conducted on producing a computer vision pipeline for this task, specifically on the cloud segmentation[2] and front identification[1] models. The BeaverCube-2 computer vision pipeline is shown in Figure 2. However, deploying software that has only been tested using ground-based computing and imaging is a risk. It is desirable to test novel algorithms in a flight-like environment before deploying them for use.

OPS-SAT is a 3U CubeSat launched on December 18, 2019; it is the first nanosatellite to be directly owned and operated by the European Space Agency (ESA). OPS-SAT is operated by the flight control team (FCT) at ESOC in Darmstadt, Germany, primarily using the Special Mission Infrastructure Lab Environment (SMILE) ground station. OPS-SAT features the Satellite Experimental Processing Platform (SEPP), an onboard computer that runs experiments from collaborators across the globe,

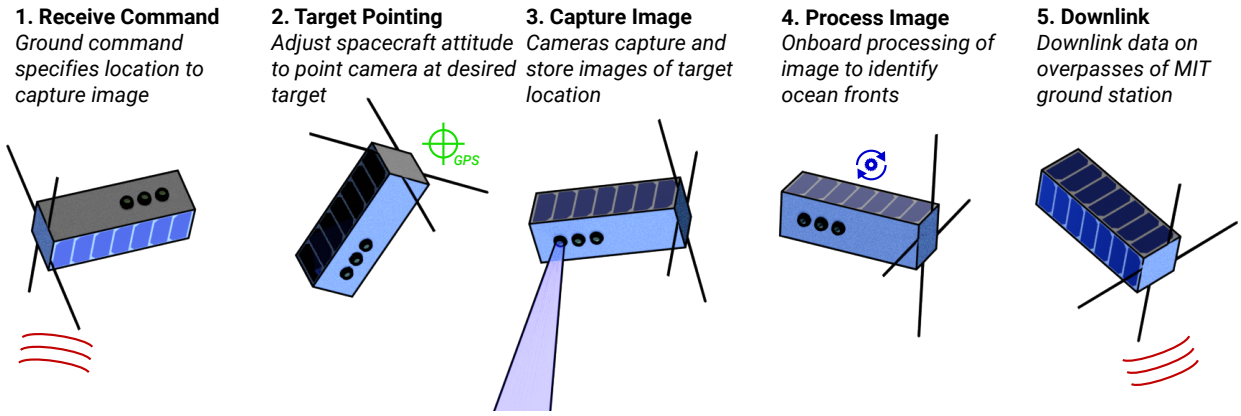


Figure 1: BeaverCube-2 concept of operations.[2]

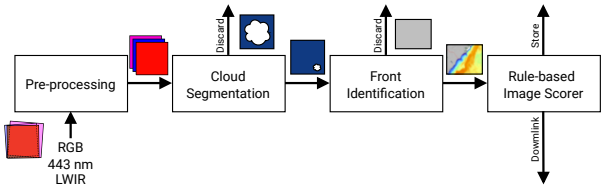


Figure 2: BeaverCube-2 image processing pipeline.

which can be used to de-risk flight software onboard a flying mission.[3] The SEPP has an Intel Cyclone V FPGA with a dual-core ARM Cortex-A9 running at 800 MHz, running a customized distribution of Yocto linux. The SEPP is much more performant than conventional space flight hardware — with sufficient onboard memory to carry out advanced software and hardware experiments[4,5,6,7] — which enables the use of modern image processing algorithms. Software for experiments is uplinked to OPS-SAT from the ground station.

One notable experiment running onboard OPS-SAT is SmartCam, an app that allows autonomous scheduling and image processing and includes a k-means algorithm for image clustering.[8] We integrated our experiment running on-orbit cloud segmentation algorithms designed for BeaverCube-2 with SmartCam, which allows us to leverage SmartCam's image pre-processing pipeline and to easily evaluate our algorithms against ESA's in-house implementation of k-means clustering for on-orbit cloud segmentation. Validating our cloud segmentation algorithms on-orbit using OPS-SAT helps us understand the challenges of image processing in an on-orbit environment and reduces risk for BeaverCube-2.

BACKGROUND

State-of-the-art cloud segmentation algorithms include rule-based methods like Fmask,[9] s2cloudless,[10] and the random forest and Bayesian thresholding algorithms used onboard the Earth Observing-1 (EO-1) mission,[11] as well as deep learning algorithms like convolutional neural networks (CNNs).[12] This work explores the implementation and on-orbit deployment of three rule-based cloud segmentation algorithms: luminosity thresholding, random forest segmentation, and k-means segmentation, as well as one deep learning algorithm – a U-Net.

Cloud Segmentation

Many state-of-the-art algorithms for generating cloud masks from satellite data are rule-based and these methods often require input data beyond RGB imagery. Two of the most popular schemes, Fmask[9] and s2cloudless,[10] used on Landsat and Sentinel-2 imagery respectively, sample from bands across the electromagnetic spectrum. Fmask uses several hand-selected thresholds to identify cloud pixels, and additionally distinguishes between land and water when identifying clouds.[9] Fmask uses 7 different Landsat bands, including a short-wave infrared (SWIR) "cirrus" band and a thermal infrared (TIRS) band.[13] Similarly, s2cloudless uses all of Sentinel-2's bands except bands 3, 6, and 7, and uses gradient-boosted random forests trained on raw data, pairwise differences between bands, pairwise ratios between bands, and averages over different bands.[10]

Another state-of-the-art mask, the continuity MODIS-VIIRS cloud mask, uses rules similar to those used for Fmask to generate cloud masks based on data from the Moderate Resolution Imaging Spectroradiometer (MODIS) and the Visible Infrared Imaging Radiometer Suite (VIIRS).[14] The continuity MODIS-VIIRS cloud mask depends on multispectral input, and uses a SWIR band to detect cirrus clouds and long-wave infrared (LWIR) bands for thermal cloud detection.[14]

Rule-based cloud segmentation methods have been tested on orbit before — EO-1 flew Bayesian thresholding and random decision forest models for on-orbit cloud detection. The EO-1 Bayesian thresholding model classified pixels individually based on their luminosity in two visible-spectrum bands and one infrared band, and the random decision forest classified pixels based on the luminosity of pixels in a 5×5 kernel surrounding the pixel of interest in three different visible-spectrum bands.[11]

Rule-based methods tend to have poor specificity and often misidentify bright ground pixels (especially snow) as clouds.[9,10] Kacker et al. demonstrated instances of Fmask misidentifying snow, coastlines, and bright rooftops of buildings as clouds.[2] Notably, many cloud masks generated with rule-based methods are considerably less accurate near the poles, likely because land near the poles tends to have a low surface temperature and is often covered in snow.[14]

Deep learning has been used for segmenting clouds in satellite imagery on the ground,[2,15] and convolutional neural networks (CNNs) in particular have shown promise for improving upon rule-based methods. Unsupervised k-means clustering has been used for image clustering onboard OPS-SAT,[16] and the resulting clusters often group images based on cloudiness. CNNs have also been used for cloud detection on OPS-SAT, but the models classified 28×28 patches as cloud or noncloud rather than classifying individual pixels, and perform with all F-scores under 0.75, possibly because these models were trained on an extremely small dataset of OPS-SAT imagery.[12] There is a gap in the literature regarding performant on-orbit cloud detection with deep learning. This paper seeks to extend prior work on cloud segmentation of satellite imagery on the ground by using deep learning to perform efficiently and accurately on orbit.

Space Considerations

For space hardware, it is generally desirable to have low size, weight, and power (SWaP). Sometimes this concept is extended to cost as well (SWaP-C). Real-time software has similarly desirable traits with analogs to the software world. In general, it is preferable for our segmentation algorithm to use minimal computational resources and power and still be fast to execute, while also maintaining a small binary size in order to be able to uplink the software. In practice, achieving all of these goals at once is often difficult, and in our experimentation we find that methods often will excel at all metrics except one.

Additionally, on-orbit pointing capability must be considered in order to match the training dataset to the model inputs. We developed our dataset with nadir-pointing images, hence it is important to avoid samples taken off-nadir, as this would not necessarily be a fair comparison and is a considerably more difficult problem, outside the scope of this work.[17]

APPROACH

Our algorithms run onboard OPS-SAT as part of the SmartCam[8] app; as such, when OPS-SAT enters specific geographic regions, SmartCam commands the spacecraft to take an image. After SmartCam crops the image, discards blurry images, and white-balances the image, our luminosity thresholding, random forest, U-Net, and k-means segmentation algorithms each segment the image and generate a cloud mask for downlink. The computer vision pipeline is detailed in Figure 3.

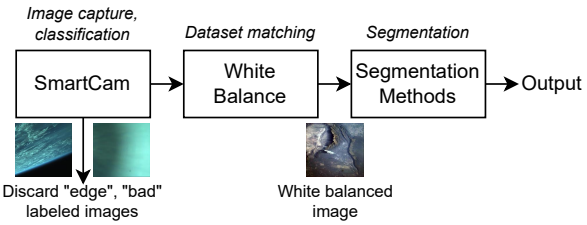
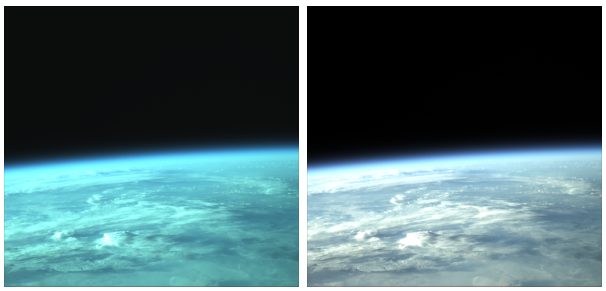


Figure 3: Computer vision pipeline for the OPS-SAT experiment.[8,16]

SmartCam

Prior to segmentation, the SmartCam app resizes images taken by OPS-SAT from their native size (2048×1944) to 614×583 thumbnails. After resizing, SmartCam uses a CNN to distinguish between images of Earth, images of the Earth limb, and blurry "bad" images. It then discards "bad" images and



(a) Raw. **(b) White-balanced.**

Figure 4: Comparison of raw and white-balanced images from OPS-SAT onboard camera. Credit: ESA.

images of the Earth limb. This form of pre-filtering helps make sure that no additional processing power is spent on attempting to segment images that are not useful in the first place. Additionally, discarding images of Earth's limb avoids issues with attempting to segment clouds off-nadir. The SmartCam can run multiple image classification models or third party executable binaries in a branchable sequence to support hyper-specialized applications of ML algorithms across an image processing pipeline.[16] The cloud detection algorithms developed for this experiment were "plugged-in" the SmartCam's image processing pipeline; demonstrating a novel application of software re-use that builds on top of another software application onboard the same flying platform. The complex challenges of cloud detection is thus decomposed into "openable" image segmentation subproblems by means of crowdsourcing into the SmartCam's pipeline.

Pre-processing

Before any segmentation, images are white balanced. White balancing is necessary to match the color distribution of the inputs to the color distribution of images in the training set, to improve performance. Images taken by OPS-SAT are significantly blue-shifted compared to conventional cameras. We reimplemented the white balancing algorithm used by the GNU Image Manipulation Program (GIMP) for use on OPS-SAT. The algorithm works very simply by stretching the 0.05^{th} and 99.95^{th} percentiles of histograms to the full range.[18] We chose this algorithm over alternate white balancing algorithms as it is simple to implement, performs well, and does not require human intervention to pick out any neutral features in an image.

Luminosity Thresholding

Luminosity thresholding, also known as Bayesian thresholding, is a simple method that classifies a pixel as cloud or non-cloud based on its luminosity in the red, green, and blue channels.[11] If a pixel is brighter than the channel threshold in all three channels, it is classified as a cloud; otherwise, it is classified as non-cloud. Luminosity thresholding is computationally efficient but is generally less accurate than more complex methods, and has flight heritage on EO-1.[11] Luminosity thresholding is equivalent to a 3-tree

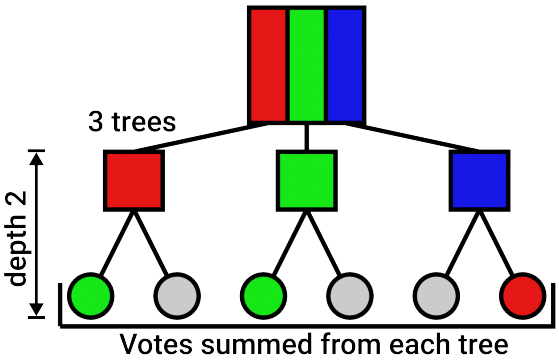


Figure 5: Luminosity thresholding architecture showing R, G, B, trees voting based on thresholds in 3-tree random forest.

random forest with trees of depth 2, as shown in Figure 5. All trees must vote unanimously in order to classify a pixel as a cloud.

Random Forest

The Earth Observing 1 (EO-1) spacecraft uses a random forest to detect clouds on orbit, and extracts features from a 5×5 kernel surrounding each pixel of interest.[11] We used the same approach in this work. As shown in Figure 6, we implement a kernel-based random forest, where each pixel is classified based on the red, green, and blue luminosities of the 3×3 window centered on the pixel being classified. We chose a 3×3 kernel over a 5×5 kernel because using a smaller window size reduces the computational complexity of the random forest algorithm, but the 3×3 kernel still preserves key features for pixel classification.

Our random forest model chose splits based on maximizing the decrease in Gini impurity, given in Equation 1.[19] In this equation, $p(x)$ represents the probability that x is a cloud pixel for x randomly selected from all pixels that reach a node N in a

random tree.

$$i_g(N) = p(x)(1 - p(x)) \tag{1}$$

Our random forest model had 10 trees, with a tree maximum depth of 18. We selected the tree maximum depth to be smaller than the number of features (27, representing the red, green, and blue luminosities of each of the 9 pixels in a 3×3 kernel) in order to avoid overfitting.[20] Our random forest architecture is shown in Figure 6.

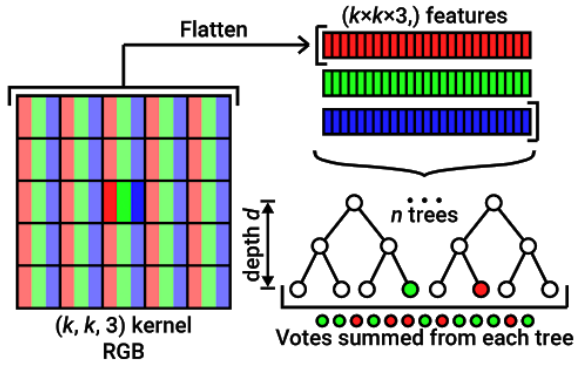


Figure 6: Random forest architecture, showing classification of a single pixel using a $k \times k$ kernel and a forest of n trees of depth d.

U-Net

Our U-Net is based on a conventional U-Net with no additional alterations, as shown in Figure 7. Focal loss — otherwise known as focal cross entropy (FCE) — is used to train our U-Net, which improves upon log loss for classifying difficult samples in a dataset.[21] Focal loss works by weighting samples based on how much of a particular classification class is present. Focal loss for the case of binary classification labels is given by Equation 2:

$$\begin{aligned} \text{FCE} = &-\alpha(1-p)^{\gamma} y \log(p) \\ &+ (1-\alpha)p^{\gamma}(1-y)\log(1-p) \end{aligned} \tag{2}$$

where y is the ground truth, p is the predicted class probability, γ is the focal parameter, and α is a weighting parameter. Higher values of γ increase the proportion of loss given to difficult samples. y and p are both bounded in the interval $[0, 1]$ in this binary classification problem.

Samples in our dataset with few clouds or mostly clouds in this case are weighted more heavily than samples with an even mix of cloud and non-cloud pixels, resulting in a model which is better tuned to accurately classify images that are mostly clouds or have only a few cloud pixels. When validating on the original dataset and comparing mean squared error (MSE) and focal loss, MSE struggles greatly on samples that are fully covered in cloud. The output appears to "zone out" and classifies cloud correctly on the edges, but classifies cloud-covered areas in the center as clear. In contrast, focal loss does a much better job in samples that are covered in clouds. While the overall accuracy does not increase significantly between MSE or FCE, prioritizing the accuracy in samples with full cloud cover is much more useful for the mission.

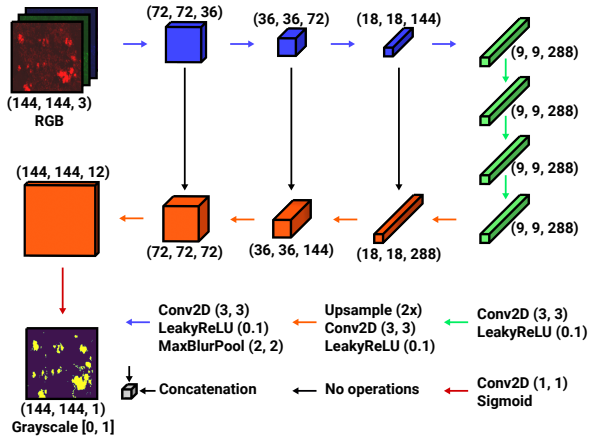


Figure 7: U-Net based architecture overview[2]

K-Means Segmentation

The k-means segmentation model implemented by the OPS-SAT FCT is a generic approach. It reads in all pixels of an image and clusters them using Lloyd's algorithm.[22] It uses `kmeans++` for initialization, which speeds up the clustering process while improving its accuracy. After clustering pixels within an image, the pixels are assigned a color for the cluster they belong to and then written as a new image file that serves as the visual output of the k-means image segmentation. Supported k-values range from 2 to 11 with $k = 2$ for cloud detection. A k limit of 11 was determined through experiment validation on the engineering model (EM) with respect to memory constraints. Predefined color palettes can be selected via a configuration file or a random palette can be generated on the fly. Figure 8 demonstrates k-means image segmentation onboard the EM using a familiar subject. Figure 9 showcases examples of on-orbit k-means image segmentation results when $k > 2$ to

demonstrate feature extraction and novelty detection capabilities beyond just cloud detection. Cloud thickness classification as well as coastline novelty detection are of particular interest in Figures 9a and 9c, respectively. Figures 10d, 11d, and 12d show how cloud detection can be configured with $k = 2$ coupled with a black and white palette configuration to generate a cloud mask as the color-coded segmented image. A ratio is calculated for the white pixel count, representing clouds, over the image's total pixel count. This ratio directly translates to a cloud coverage percentage which is used to filter against thresholds that group the acquired images into buckets of cloudiness. The FCT can then automate which images to downlink based on cloud coverage thus delegating decision-making to in-space autonomous operations.

(a) Before segmentation. (b) After segmentation.

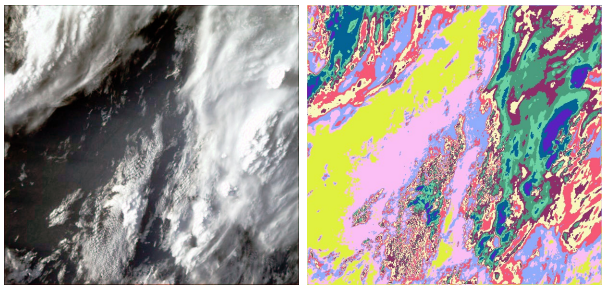
Figure 8: Picture of flight control team member Vladimir during k-means validation on the EM ($k = 5$ with the `Set3` palette).

IMPLEMENTATION

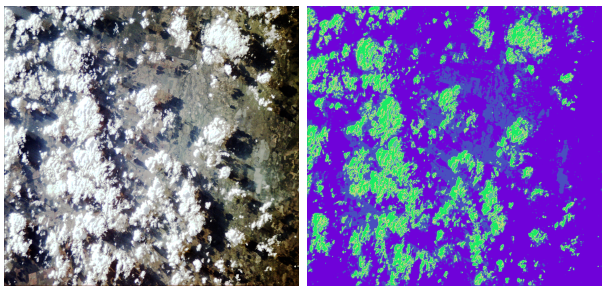
The implemented algorithms were driven by OPS-SAT's recommended 10 MB uplink limit and the libraries available on the SEPP. We were able to use Tensorflow Lite to deploy our U-Net model, but our other models were implemented in `C++` and cross-compiled to run on the SEPP.

Luminosity Thresholding

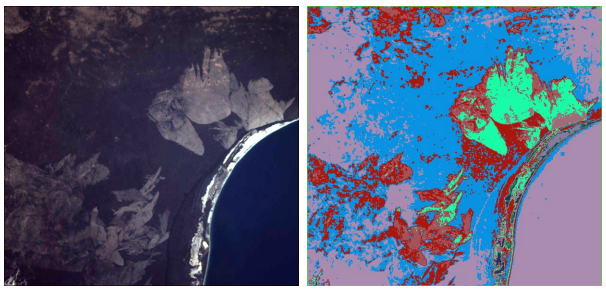
We implemented luminosity thresholding ourselves in `C++`. Because luminosity thresholding is computationally simple and requires only enough memory to load an image for segmentation, it was straightforward to port luminosity thresholding algorithms to work on embedded hardware.



(a) Cumulus clouds ($k = 10$, $t = 0m\,12.41s$)



(b) Stratos clouds ($k = 5$, $t = 2m\,35.90s$)



(c) Clear skies with coastline ($k = 10$, $t = 0m18.29s$)

Figure 9: On-orbit k-means image segmentation results with $k > 2$ and randomly generated color palettes for cumulus clouds, stratos clouds, and clear skies with coastline. The t variable is the execution time of the k-means algorithm to segment the acquired images in memory and write the result as a new image file. Credit: ESA.

Random Forest

We used the open-source Ranger library, implemented in `C++`, to train our random forest model.[23] Ranger is designed to operate on textual data and is not designed for embedded hardware, so we modified the Ranger library to train on images and generate cloud masks.

Random forest models trained with Ranger cannot be easily transferred between computers with

different instruction sets, so we trained our random forest model on a Raspberry Pi in order to match the ARM32 environment of OPS-SAT's SEPP. We trained our random forest model on only a subset of our Landsat dataset containing 5 images from each category because of the computational and memory constraints posed by training on embedded hardware.[2]

OPS-SAT's recommended 10 MB uplink limit drove us to make disk footprint-saving architectural decisions when implementing our random forest model. We implemented a new memory-saving mode to persist Ranger model files, representing training data as integers, in order to reduce the size of our random forest model. We also reduced the number of trees in our random forest from 25 trees to 10 trees in order to shrink the resulting random forest model.

The final Ranger binary used in this work is 4.9 MB and the final random forest model is 12 MB. Although the final random forest model is larger than 10 MB, compressing our models before uplink allows us to satisfy the recommended uplink size requirements.

U-Net

The U-Net model is initially trained on TensorFlow and converted to a TensorFlow Lite (tflite) model using fully integer optimizations. The final size of the tflite model is 3.3 MB. The tflite inferencer provided executes the model on-orbit, which adds approximately another 3 MB to the file size.

As the tflite model cannot take variable size inputs, the input image is split up into separate patches and fed into the model sequentially.

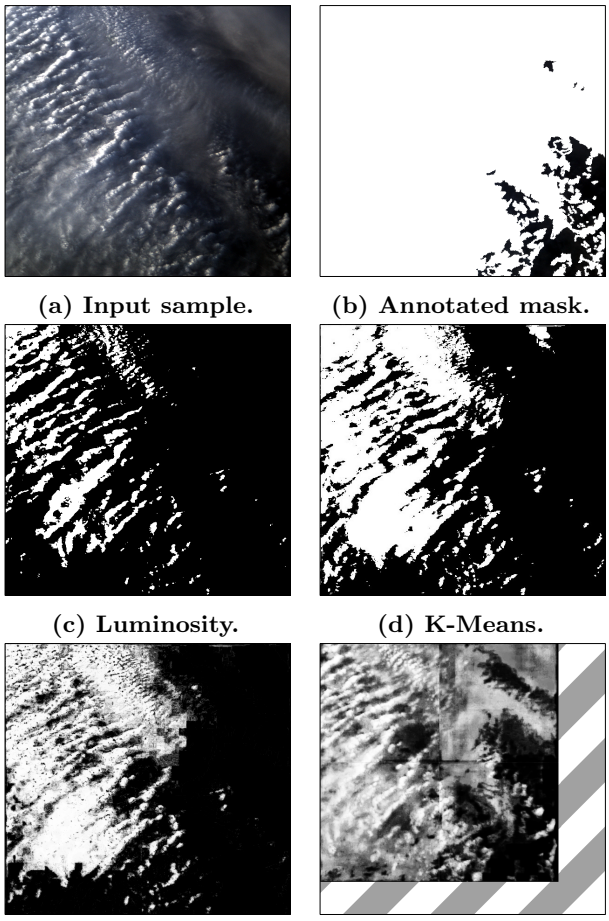
K-Means Segmentation

We relied upon the version of k-means segmentation implemented by the OPS-SAT FCT, which is implemented in `C++` using the dkm k-means clustering library.[16] Because k-means segmentation was already implemented by the OPS-SAT FCT and installed on OPS-SAT, the size of the k-means segmentation binaries did not contribute to the size of our uplinked experiment.

RESULTS

A radiation induced anomaly which corrupted OPS-SAT's filesystem resulted in a delay in deploying our random forest model and we were unable to run the latter on-orbit. Instead, we took the images that our other three models segmented onboard OPS-SAT and processed them with our random forest model on the OPS-SAT engineering model, which has identical hardware to the flight model. We then generated cloud masks for these images by hand in order to quantitatively evaluate the performance of our algorithms.

Comparative Analysis



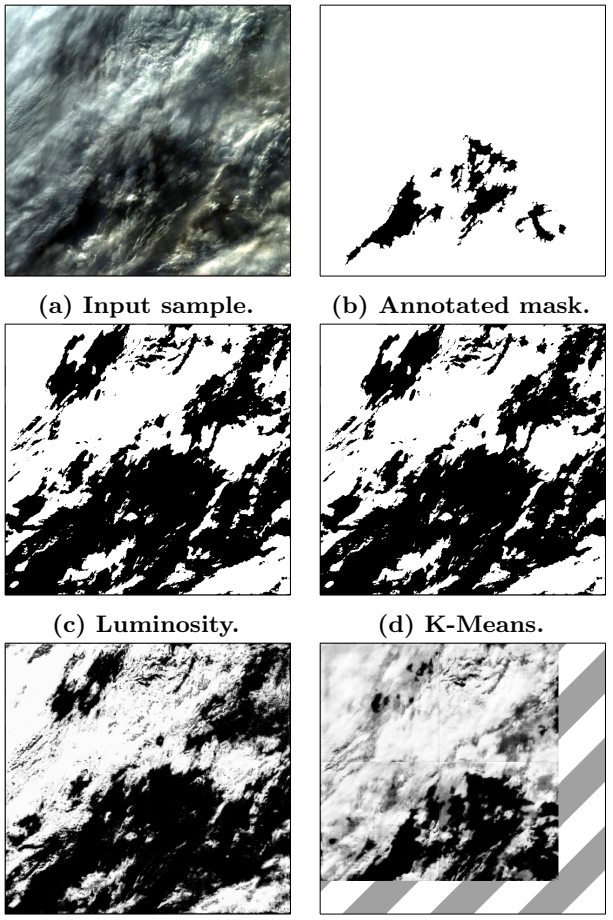
(a) Input sample. **(b) Annotated mask.**

(c) Luminosity. **(d) K-Means.**

(e) Random forest. **(f) U-Net.**

Figure 10: Comparison of the output cloud mask from each method on cloudy off-nadir white-balanced sample input.

Figure 10, Figure 11, and Figure 12 show a comparison of the segmentation maps from every method on a white balanced inputs. Figure 10 and Figure 11 show the results on cloudy samples, and Figure 12 shows outputs from an "easy" sample, as the clouds look distinct from the background and an ideal single-dimensional classifier such as k-means segmentation

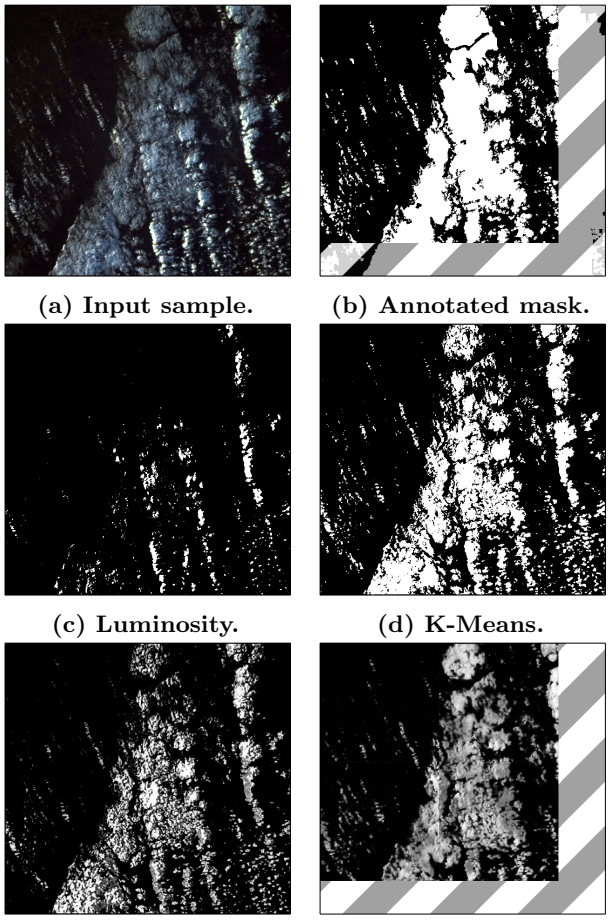


(a) Input sample. (b) Annotated mask.

(c) Luminosity. (d) K-Means.

(e) Random forest. (f) U-Net.

Figure 11: Comparison of the output cloud mask from each method on cloudy white-balanced sample input.



(a) Input sample. (b) Annotated mask.

(c) Luminosity. (d) K-Means.

(e) Random forest. (f) U-Net.

Figure 12: Comparison of the output cloud mask from each method on high contrast white-balanced sample input over ocean. Mask is annotated up to 512 px width and height to match U-Net output.

can segment them easily. Insufficient samples were taken to draw conclusions about dataset-wide metrics, so these extreme samples were picked to compare each method against.

Qualitatively, it can be observed that the U-Net performs the best overall, with the other methods failing to segment a significant portion of the clouds in the image. Seams can be seen in the U-Net output based on where images are spliced back together after being processed in individual patches. Table 4 shows a comparison of the runtimes of each of the models. While the U-Net is much more accurate overall, its runtime is unfortunately much higher than those of the other two methods, motivating the use of the AI accelerator SoC on BeaverCube-2. Additionally, the U-Net output is limited to 512 px in width and height due to an algorithm design decision with respect to patching dimensions. For this reason, comparisons in all metrics are made on 512 px crops of all the images.

The evaluated metrics are:

$$\text{Accuracy} = \frac{\Sigma\,[\text{TP} + \text{TN}]}{\Sigma\,[\text{TP} + \text{TN} + \text{FP} + \text{FN}]} \tag{3}$$

$$\text{Sensitivity} = \text{Recall} = \frac{\Sigma\,\text{TP}}{\Sigma\,[\text{TP} + \text{FN}]} \tag{4}$$

$$\text{Specificity} = \frac{\Sigma\,\text{TN}}{\Sigma\,[\text{TN} + \text{FP}]} \tag{5}$$

$$\text{Balanced Accuracy} = \frac{\text{Sensitivity+Specificity}}{2} \tag{6}$$

$$\text{Precision} = \frac{\Sigma\ \text{TP}}{\Sigma\ [\text{TP} + \text{FP}]} \tag{7}$$

$$\text{F}_1\ \text{Score} = \frac{\Sigma\ \text{TP}}{\Sigma[\text{TP}\ + \frac{1}{2}[\text{FP} + \text{FN}]]} \tag{8}$$

where TP are true positives, TN are true negatives, FP are false positives, and FN are false negatives.

Full results for each method are shown in Tables 1, 2, and 3. All methods show very high specificity and comparatively lower sensitivity, which leads to many false negatives in segmented images.

The U-Net model performs best over all images, with balanced accuracies ranging from 77%-89%, and an F_1 score of 0.69 to 0.87. In particular, the U-Net retains a high F_1 scores over cloudy samples, which is the situation in which it has the most value. The k-means model performs best overall in the easy sample as shown in Table 3, as the two classes are equally distributed in the data, and the ideal separator can be easily found. In contrast, the luminosity-based classifier, which can be thought of as an instance of k-means with a fixed separator, performs worse on the easy sample as it optimizes for the ideal separator over the training dataset rather than that over the sample.

Resource Utilization

The CPU usage on both cores of the SEPP are shown in Figures 13, 14, and the memory required during the experiment run is shown in Figure 15.

Figures 13, 14 show most of the CPU usage is on CPU-0, with a small amount of usage on CPU-1. This shows that the experiment is multi threaded, and gains in performance can be made with efficient parallelization. The algorithms presented in this paper are trivially parallelizable, and there are many avenues for pursuign increased performance.

Figure 15 shows memory usage during the epxeriment. Overall, peak memory usage for the luminosity thresholding and k-means methods are very small due to the only requirement being to store the image in memory, and cannot be seen clearly in the plot. On the otherh and, the U-Net and random forest methods can clearly be seen as pleateaus of memory usage one after each other. The U-net uses 90 MB of memory at its peak, whereas the random forest uses 170 MB of memory at its peak. The increased memory usage of random forest is due to the fact that it is a data driven method and does not generate a model, but uses an ensemble of branches of the original training data.

Figure 16 shows the current draw during the experiment. At peak usage, the experiment drew an increase of 140 mA of current compared to the baseline current draw. The temperature rise generated from the power draw is shown in Figure 17, resulting in a peak temperature rise of 1.8 °C for the second temperature sensor on the SEPP.

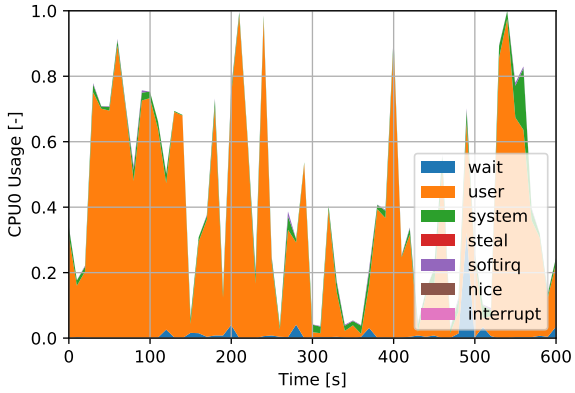


Figure 13: CPU-0 usage during SEPP run of classifying a single image.

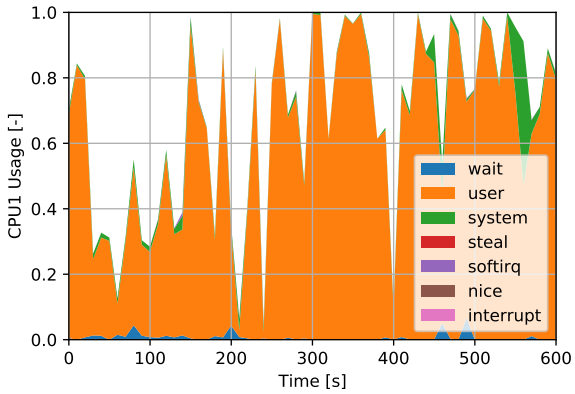


Figure 14: CPU-1 usage during SEPP run of classifying a single image.

CONCLUSION

Running our algorithms onboard OPS-SAT significantly helped us de-risk our algorithms in preparation of on-orbit usage in BeaverCube-2. The process allowed us to accurately gauge the advantages and disadvantages of all models. In particular, a large tradeoff in model size and processing time is observed

Table 1: Metrics evaluated over cloudy, off-nadir sample in Figure 10.

Method	Accuracy	Balanced Accuracy	Sensitivity	Specificity	Precision	Recall	F_1 Score
Luminosity	0.22	0.56	0.12	1.00	1.00	0.12	0.21
K-Means	0.40	0.66	0.32	1.00	1.00	0.32	0.49
Random Forest	0.39	0.66	0.31	1.00	1.00	0.31	0.48
U-Net	0.62	0.78	0.57	1.00	1.00	0.57	0.73

Table 2: Metrics evaluated over cloudy sample in Figure 11.

Method	Accuracy	Balanced Accuracy	Sensitivity	Specificity	Precision	Recall	F_1 Score
Luminosity	0.60	0.78	0.57	1.00	1.00	0.57	0.73
K-Means	0.51	0.74	0.47	1.00	1.00	0.47	0.64
Random Forest	0.64	0.80	0.60	1.00	1.00	0.60	0.75
U-Net	0.79	0.89	0.77	1.00	1.00	0.77	0.87

Table 3: Metrics evaluated over easy sample in Figure 12.

Method	Accuracy	Balanced Accuracy	Sensitivity	Specificity	Precision	Recall	F_1 Score
Luminosity	0.70	0.54	0.07	1.00	1.00	0.07	0.13
K-Means	0.89	0.84	0.68	1.00	0.99	0.68	0.81
Random Forest	0.80	0.69	0.38	1.00	1.00	0.38	0.55
U-Net	0.85	0.77	0.54	1.00	0.99	0.54	0.69

Table 4: Run time of all tested methods.

Model	Run time
Luminosity	2s
K-Means	3s
Random Forest	1m44s
U-Net	7m13s

when using purely data-driven models compared to neural networks or simpler statistical models.

The only model that performs accurately enough in scenes with complete cloud cover is the U-Net; however, it has a runtime of over seven minutes on the SEPP. The simpler statistical models such as the k-means clustering perform the worst in these samples as they can only look at differences within the sample, and cannot accurately segment samples where one class dominates the image. While the model takes over seven minutes to run on the SEPP, the AI accelerator SoC on BeaverCube-2 is highly customizable, and so we expect that we can achieve a significant speedup by adapting it to efficiently run our U-Net.

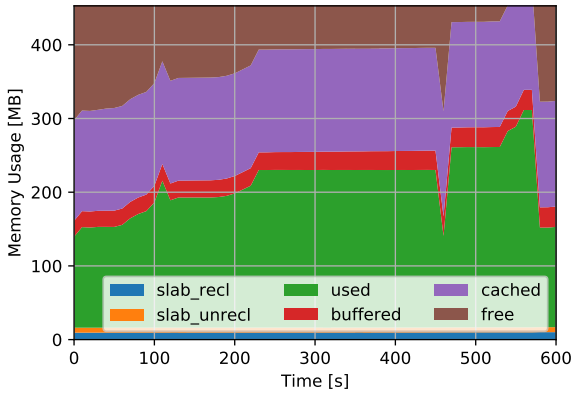


Figure 15: Memory usage during SEPP run of classifying a single image.

Additionally, deployment on OPS-SAT helped to identify clear parts of the computer vision pipeline that were missing, in particular white balancing. The models can now be ported to our AI accelerator SoC. Our luminosity thresholding model and white bal-

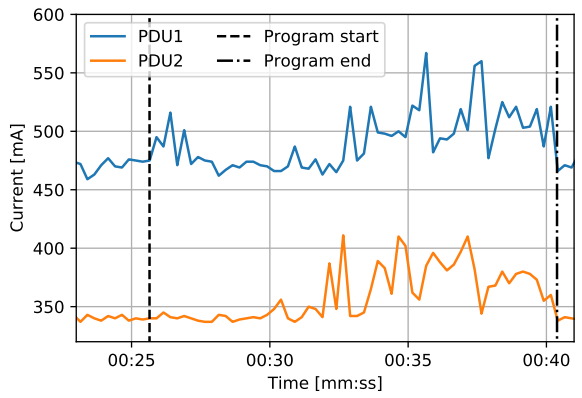


Figure 16: Current provided by both power distribution units (PDUs) on the SEPP during experiment operation.

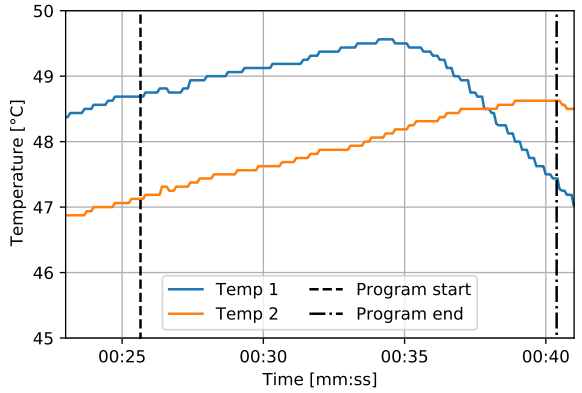


Figure 17: Temperature on two thermal sensors on SEPP during experiment operation.

ancing module are written in vanilla `C++` and are easily transferable. However, we expect some challenges adapting our more complex models, particularly the U-Net, to take full advantage of the efficient parallelization offered by the architecture of our AI accelerator SoC.

FUTURE WORK

In anticipation of BeaverCube-2's launch in 2023, we plan to continue improving the accuracy of our cloud segmentation algorithms while also working to port our models to BeaverCube-2's hardware.

Model Improvements

Although our U-Net model is translationally equivariant, it is not rotationally equivariant; its feature maps are not equivariant to rotations of a cloud or the whole image about the axis of the spacecraft's camera. We are currently developing an $SE(2)$-equivariant steerable CNN, which will be equivariant to cloud and camera rotations.[24] Rotationally equivariant steerable CNNs require fewer parameters than traditional CNNs and generalize better over orientation for rotation-equivariant classification tasks.[25] As such, we expect that our $SE(2)$-equivariant steerable CNN will outperform our U-Net model in generalizing from our limited training dataset and will ultimately classify clouds more accurately than the U-Net.

Rotational equivariance achieved through steerable filter CNNs has already led to improvements in segmenting medical imagery.[26] We expect that a rotationally equivariant model will better harness orbital geometry and symmetry and improve the consistency and accuracy of our models.

Runtime performance of our models can be improved by tailoring the models to OPS-SAT's available hardware. OPS-SAT has an FPGA component that can be used for hardware acceleration of models, but usage was outside the scope of this work. Other experiments on OPS-SAT have used the FPGA to improve the performance of cloud segmentation algorithms.[12] Additionally, all of the methods presented are trivially parallelizable and could utilize both of the cores of the SEPP CPU resulting in significant performance improvements.

Deployment to BeaverCube-2

Our models are designed to run on OPS-SAT's SEPP, but BeaverCube-2 uses a different AI Computational Accelerator SoC. As a result, we will need to make some changes to our model implementations in order to best harness BeaverCube-2's hardware. In particular, code for luminosity thresholding and random forest methods will need to be modified to make use of the additional available hardware onboard in order to speed up and parallelize the methods. The U-Net model can be used as is, since a workflow exists for integration of TensorFlow models onto the AI accelerator.

We will be installing our initial models on BeaverCube-2 prior to flight rather than uplinking them, and so our model binaries can be larger. We will explore the costs and benefits of training a random forest model with more than 10 trees, since for BeaverCube-2 the size of the pre-initialized model is not subject to uplink size constraints.

Landsat imagery proved to be a useful analog to orbital imagery taken on OPS-SAT after white balancing; however, some improvements can still be made to better align the weights with the expected inputs. A few-shot learning method with on-orbit training could be used to transfer learn weights and improve model performance. This would be an unsupervised method and transductive transfer learning techniques would need to be used.[27]

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. 2141064.

This material is also based upon work sponsored by the Northrop Grumman Corporation as part of BeaverCube-2 hardware development.

The authors would like to thank the OPS-SAT flight control team, especially Vladimir Zelenevskiy, for help and advice on our experiment.

REFERENCES

[1] Violet C. Felt. Machine Learning Models for On-Orbit Detection of Temperature and Chlorophyll Ocean Fronts. Master's thesis, Massachusetts Institute of Technology, 2022.

[2] Shreeyam Kacker, Alex Meredith, Joe Kusters, Hannah Tomio, Kerri Cahoy, and Violet Felt. On-orbit rule-based and deep learning image segmentation strategies. In *AIAA SCITECH 2022 Forum*, page 0646, 2022.

[3] Georges Labrèche, David Evans, Dominik Marszk, Tom Mladenov, Vasundhara Shiradhonkar, and Vladimir Zelenevskiy. Agile Development and Rapid Prototyping in a Flying Mission with Open Source Software Reuse On-Board the OPS-SAT Spacecraft. In *AIAA SCITECH 2022 Forum*, page 0648, 2022.

[4] David Evans, Georges Labrèche, Tom Mladenov, Dominik Marszk, Vladimir Zelenevskiy, and Vasundhara Shiradhonkar. OPS-SAT LEOP and Commissioning: Running a Nanosatellite Project in a Space Agency Context. In *Proceedings of the 36th Annual Small Satellite Conference*, Logan, UT, 2022. SmallSat Session IX: Recent Launches, SSC22-IX-06, AIAA/USU.

[5] David Evans, Georges Labrèche, Dominik Marszk, Sam Bammens, Miguel Hernández-Cabronero, Vladimir Zelenevskiy, Vasundhara Shiradhonkar, Milenko Starcik, and Maximilian Henkel. Implementing the New CCSDS Housekeeping Data Compression Standard 124.0-B-1 (based on POCKET+) on OPS-SAT-1. In *Proceedings of the 36th Annual Small Satellite Conference*, Logan, UT, 2022. SmallSat Technical Session XII: Communications, SSC22-XII-03, AIAA/USU.

[6] Dominik Marszk and David Evans and Tom Mladenov and Georges Labrèche and Vladimir Zelenevskiy and Vasundhara Shiradhonkar. MO Services and CFDP in Action on OPS-SAT. In *Proceedings of the 36th Annual Small Satellite Conference*, Logan, UT, 2022. Weekend Sesssion IV: Advanced Concepts - Research Academia II, SSC22-WKIV-05, AIAA/USU.

[7] Tom Mladenov, David Evans, and Vladimir Zelenevskiy. Implementation of a gnu radio-based search and rescue receiver on esa's ops-sat space lab. *IEEE Aerospace and Electronic Systems Magazine*, 37(5):4–12, 2022.

[8] Georges Labrèche, David Evans, Dominik Marszk, Tom Mladenov, Vasundhara Shiradhonkar, and Vladimir Zelenevskiy. Artificial Intelligence for Autonomous Planning and Scheduling of Image Acquisition with the SmartCam App On-Board the OPS-SAT Spacecraft. In *AIAA SCITECH 2022 Forum*, page 2508, 2022.

[9] Zhe Zhu and Curtis E Woodcock. Object-based cloud and cloud shadow detection in Landsat imagery. *Remote sensing of environment*, 118:83–94, 2012.

[10] Anze Zupanc. Improving cloud detection with machine learning, 2017.

[11] Kiri L Wagstaff, Alphan Altinok, Steve A Chien, Umaa Rebbapragada, Steve R Schaffer, David R Thompson, and Daniel Q Tran. Cloud filtering and novelty detection using onboard machine learning for the EO-1 spacecraft. In *Proc. IJCAI Workshop AI in the Oceans and Space*, 2017.

[12] Frédéric Férésin, Erwann Kervennic, Yves Bobichon, Edgar Lemaire, Nassim Abderrahmane, Gaétan Bahl, Ingrid Grenet, Matthieu Moretti, and Michael Benguigui. In space image processing using AI embedded on system on module: example of OPS-SAT cloud segmentation. In *2nd European Workshop on On-Board Data Processing*, 2021.

[13] Zhe Zhu, Shixiong Wang, and Curtis E Woodcock. Improvement and expansion of the Fmask algorithm: Cloud, cloud shadow, and snow detection for Landsats 4–7, 8, and Sentinel 2 images. *Remote sensing of Environment*, 159:269–277, 2015.

[14] Richard A Frey, Steven A Ackerman, Robert E Holz, Steven Dutcher, and Zach Griffith. The continuity MODIS-VIIRS cloud mask. *Remote Sensing*, 12(20):3334, 2020.

[15] Zhiwei Li, Huanfeng Shen, Qing Cheng, Yuhao Liu, Shucheng You, and Zongyi He. Deep learning based cloud detection for medium and high resolution remote sensing images of different sensors. *ISPRS Journal of Photogrammetry and Remote Sensing*, 150:197–212, 2019.

[16] Georges Labrèche, David Evans, Dominik Marszk, Tom Mladenov, Vasundhara Shiradhonkar, Tanguy Soto, and Vladimir Zelenevskiy. OPSSAT Spacecraft Autonomy with TensorFlow Lite, Unsupervised Learning, and Online Machine Learning. In *2022 IEEE Aerospace Conference*, 2022.

[17] Hanxiang Hao, Sriram Baireddy, Kevin LaTourette, Latisha Konz, Moses Chan, Mary L Comer, and Edward J Delp. Improving Building Segmentation for Off-Nadir Satellite Imagery. *arXiv preprint arXiv:2109.03961*, 2021.

[18] White Balance. `https://docs.gimp.org/en/gimp-layer-white-balance.html`. Accessed: 2022-06-01.

[19] Gilles Louppe. Understanding random forests: From theory to practice. *arXiv preprint arXiv:1407.7502*, 2014.

[20] Stefan Wager, Trevor Hastie, and Bradley Efron. Confidence intervals for random forests: The jackknife and the infinitesimal jackknife. *The Journal of Machine Learning Research*, 15(1):1625–1651, 2014.

[21] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection.

[22] Stuart Lloyd. Least squares quantization in PCM. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[23] Marvin N Wright and Andreas Ziegler. ranger: A fast implementation of random forests for high dimensional data in C++ and R. *arXiv preprint arXiv:1508.04409*, 2015.

[24] Gabriele Cesa. E (2)-Equivariant Steerable CNNs. 2020.

[25] Maurice Weiler, Fred A Hamprecht, and Martin Storath. Learning Steerable Filters for Rotation Equivariant CNNs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 849–858, 2018.

[26] Simon Graham, David Epstein, and Nasir Rajpoot. Dense steerable filter cnns for exploiting rotational symmetry in histology images. *IEEE Transactions on Medical Imaging*, 39(12):4124–4136, 2020.

[27] Marcus Rohrbach, Sandra Ebert, and Bernt Schiele. Transfer learning in a transductive setting.