Implementation

1. Dataset loading pipeline

As mentioned in section 2.3, the dataset is split into 18 training and 20 test images. Each full scene is stored as a 16-bit unsigned integer .TIF file with four channels: RGB and NIR (TODO maybe add spectral range in nm). These images have a resolution of around 8000x8000 pixels and a size of approximately 130 MB. To reduce memory load and optimize training, validation and testing performance, the dataset is preprocessed by cropping each scene into smaller patches of 384x384 pixels.

Unlike typical RGB images used in computer vision, the channels are not combined. Instead, each channel is stored in its respective directory for both training and testing subsets. For visualization purposes, the full scene images are rendered using a standard false-color composite in which the NIR band is mapped to red, red to green, and green to blue. This mapping coresponds to the commonly used Color Infrared (CIR) visualization in remote sensing. CIR imagery enhances features such as vegetation and clouds, which exhibit distinct reflectance patterns across the visible and NIR spectrum.

However, these false-color visualizations were not used as input for the CNN during training, validation, or inference. They served solely as visual references for qualitative insprection by the human observer.

TODO check patches and scenes to verify what bands were replaced

TODO provide 1 scene false color image and rgb nir gt patches as example images

The following folder structure of the dataset was used:

TODO provide folder structure

Additionally .csv files containing lists of patch filenames and corresponding scene IDs were provided and leveraged for efficient dataset construction using TensorFlow data pipeline utilites.

One notable detail is that, due to the cropping and padding of border patches (in order to achieve standard patch size 384x384) and the tilted geometry of Landsat 8 imagery, some resulting patches contain no meaningful information – appearing completelly black across all four channels. These empty patches were excluded from training and validation to avoid introducing noise or misleading the model.

*As mentioned in section 2.3 the dataset is split into 18 training and 20 test images. Each entire scene is a .TIF 16 bit uint 4 channel (RGB and NIR <maybe add spectral range in nm?>) image with size around 8000x8000 pixels. Weighing around 130MB each it requeres preprocessing in order to lighten and maximize the effectiveness of training, validation and testing. The dataset is already prepared by cropping each scene into 384x384 pixels patches. Each one out of 4 channels is represented separately in training as well as in testing subsets. ГUnlike other computer vision images, these channels are not combined together. Instead, they are in their correspondig directories. For additional visualization purposes the entire scenes are represented using standard false-color composite, where the NIR band was mapped to red, red to green, green to blue. This is consistent with traditional false-color visualizations in remote sensing, commonly reffered to as Color Infrared (CIR). Such composites are particularly useful for distinguishing vegetation and clouds, which reflects differently across the visible and NIR spectrum. However, the natural false color entire scenes were not used by CNN at any time during training, validation and testing. They served more as visual confirmation for human eye results checking. <TODO check patches and scenes to verify what bands were replaced>.*

*<Provide 1 entire scene false color image and r,g,b,nir,gt patches as example images>*

*Following folder structure of the dataset was exploited <provide folder sturture>*

*.csv files containing names of training and testing patches and sceneids were provided and used for efficient dataset building with TensorFlow methods.*

*An important detail to mention is, that after cropping and padding border patches in order to stadartise 384x384 pixel patches as well as due to tilted images in original Landsat 8 scenes a certain amount of patches contains no information and is therefor pitch black in all 4 channels. These patches havent participated in training and validation.*

* 1. Train and validation subsets

To build a training and validation pipeline, the function BuildDS was implemented along with several supporting helper functions, all included in the load.py file. Later on, the function was extended to optionally load the test dataset as well – this will be discussed separately in !!!Test subset!!!

For efficient data handling and memory management, built-in TensorFlow utilities (referred to as tf in the following text and code) were used. Specifically, tf.Data.TextLineDataset() was used to read the .csv file line by line – each line representing the filename of a patch – which served as the foundation for the dataset pipeline.

The complete training subset originally contains 8400 patches. However, as noted earlier in !!!chapter!!!, only 5155 of them actually contain valid data and therefore used for training and validation. The dataset is shuffled and then split into training and validation subsets. The ration between the two can be configured as needed.

Using the .map methon, each text line is first expanded into five full paths: the corresponding red, green, blue, NIR and ground truth patch filenames. Each of these file paths is then replaced by ist actual image content, loaded as tensor. This transformation is implemented in the helper function loadDS, which itself calls another utility, loadTIF. At this stage, each dataset element is a tuple of TensorFlow tensors representing the input image and its corresponding ground truth.

The loadDS together with loadTIF function are loading each .TIF image and performing the necessary conversions:

* RGB and NIR patches, originally stored as 16-bit unsigned integers (range 0 - 65535), are cast to float32 and normalized to the range 0-1
* Ground truth masks, which originally contain values of either 0 or 255 (as uint8), are binarizes to values of 0 and 1, and also cast to float32.

An additional feature of the loadDS function allows optional resizing (downsampling) of the input images, if a target image size is provided. Importantly, the image loading and transormation pipeline ensures no information loss up to the point of resizing, where a reduction in resolution is intentional and controlled.

As final preparation steps, the dataset is: shuffled, batched, prefetched, and set to repeat indefinitely (cycled).

*In order to build training a validation pipeline the function buildDS was implemented together with the correspondent helper functions altogether provided in load.py file. Later on the optional loading af test dataset was builded in in the same function which will be further discussed in [Test subset]*

*For an effective data flow and for the sake of memory the TensorFlow (in further code snippets and mentionings reffered to as tf) built in utilites were take advantage of. With the help of a package function tf.data.TextLineDataset() the lines of .csv file – that are representing patches filenames – were taken as basis for future dataset pipeline. The full training subset size provided by default comprises to 8400 patches in total. However, as mentioned earlier in <>, only 5155 of them actually contain useful information for training and validation. The respective yet textline dataset is shuffled and splitted furthermore into test and validation parts. The ratio test/val can be varied. At this point each object of the dataset is a TensorFlow instance, utilizing the .map method (or feature) each textline is exanded first into 5 full paths to the corresponding red, green, blue, nir and ground truth patches. And then each replaced with tensor arrays representing loaded .TIF images. Additionaly in order to implement this the helper function loadDS together with loadTIF helper function were implemented. Feel free to investigate the functionality of them in the provided code. Important note:*

*While loading the RGB and NIR patches get ultimately casted from 0 to 65535 16 bit unsigned integer into 0 to 1 range float32. Whereat ground thruth mask originally containing only 0 or 255 unsignet 8 bit integer values gets casted into binary 0 or 1 format but represented in float32 too. Furthermore loadDS function posesses the possibility to resize (actually downsize) the image to load if needed and target image size provided. It has to be mentioned that it is guaranteed up until resizing (where its inevitable) there is no information loss while loading and transforming images. As final steps in preparation the dataset is shuffled, batched, prefetched and infinitely cycled.*

* 1. Test subset

Background

1. State-of-the-art CNNs for Cloud Image Segmentation

CNNs are a class of neural networks that apply small, learnable filters – known as convolutional kernels – across an image to extract spatial features. A CNN typically consists of multiple convolutional layers stacked sequentionally. Each layer applies a set of filters that capture different visual patterns, such as edges or textures. As the network goes deeper, the spatial resolution of the image decreases, while the depth (i.e., number of channels) increases. This is a result of applying multiple filters and optionally using pooling layers, which downsample feature maps to reduce computational complexity and introduce spatial invariance.

In image segmentation tasks such as cloud detection, preserving spatial resolution is critical. Therefore, architectures often include an upsampling mechanism to reconstruct high-resolution output from compressed feature representations. This is achieved through transposed convolution (also known as deconvolution). While standard convolution reduces spatial resolution by aggregating local pixel values, transposed convolution performs the reverse: it distributes each value in the smaller feature map across a larger output, effectively increasing spatial dimensions and reversing the compression.

One of the most influential architectures for image segmenation is the U-Net. Originally introduced by Ronneberger et al [\*\*\*] for biomedical image segmentation, U-Net has become a standard in many domains, including remote sensing and cloud detection. A U-Net consists of an encoder-decoder structure. The encoder compresses the input image spatially while increasing its feature dimensionality. The decoder then reconstructs the spatial dimensions, progressively reducing the number of channels. The U-Net architecture shown in the diagram 1 was used by Mohajerani et al. [\*\*\*] for their cloud detection algorithm.

A key innovation in U-Net is the use of skip connections, which dierectly link feature maps from the encoder to corresponding layers in the decoder with the same spatial size. These connections preserve fine-grained spatial details and significantly enhance segmentation quality. Moreover, they mitigate the vanishing gradient problem, facilitating the training of deeper networks and improving convergence.

1. Google Coral Dev Board Mini and Edge TPU

Running machine learning inference on embedded systems is referred to as edge inference. The Coral dev Board Mini, developed by Google, is a compact single-board computer designed for such edge AI applications. It features quad-core MediaTek 8167s System-on-a-Chip (SoC) on the Armv8-A architecture, along with a dedicated Edge TPU – a hardware accelerator optimized for executing TensorFlow Lite models using 8-bit integer operations.

The Edge TPU delivers up to 4 trillion operations per second (TOPS) of performance while consuming only around 2 watts of power, making it ideal for use in resource-constrained environments such as satellites, where energy efficiency and reliability are crucial.

To deploy a model on Edge TPU, it must be:

* Converted into the TensorFlow Lite (.tflite) format,
* And fully quantized to 8-bit integers.

These requirements directly affect model architecture, training strategy, and tooling. For instance, certain operations unsopported by Edge TPU must be avoided, and quantization aware training or post-training quantization must be considered early in the development process.

The following image from Coral documentation summarizes the model conversion and deployment workflow.

1. 38-Cloud Landsat 8 dataset

Landsat 8 is an Earth observation satellite launched on February 11, 2013, providing high-resolution multispectral imagery, including visible, near infrared (NIR), and thermal infrared bands. For cloud segmentation tasks, the red, green, blue (RGB) and NIR channels are particularly valueable due to their ability to capture both visual and athmospheric information.

This thesis utilizes a dataset consisting of 38 annotated satellite images from the Landsat 8 mission, commonly referred to as the 38-Cloud dataset. [\*\*\*]. The dataset has been introduced and adapted in the following scientific publications [\*\*\*]. Each image contains four spectral channels (RGB and NIR), along with a manually annotated dround truth mask that labels cloud pixels at the pixel level.

The dataset is divided into a training set containing 18 scenes and a test set with the remaining 20 scenes.

*Landsat 8 is a satellite mission launched in 2013 that provides high-resolution multispectral imagery, which includes visible, near-infrared, and thermal bands. [\*\*\*] For cloud segmentation, the red, green, blue (RGB) and near-infrared channels are paricularly useful.*

*In this thesis, a dataset consisting of 38 cloud-annotated images from the Landsat 8 mission is used. The dataset was presented and adjusted in the following scientific papers. Each image includes 4 channels – RGB and NIR – as well as the corresponding manually extracted pixel-level ground truth. The dataset is split into training part containing 18 sceneids and testing part containing respectively 20 sceneids.*

Motivation

Electronic components in satellites and spacecraft are exposed to intense radiation, high-energy particles, and extreme temperature fluctuations. To ensure reliability under these harsh conditions, manufacturers traditionally use certified space-grade components. However, design, production, and certification of such components are costly and time-consuming.

The AITHER project aims to investigate whether COTS (Commercial Off-The-Shelf) components can be effectively used in mano- and microsatellites (10-100 kg). Specifically, the project explores the feasibility of using COTS-based onboard architecture to perform reliable and computationally demanding tasks, such as matrix operations, directly in space.

As part of this initiative, the 10kg nanosatellite OOV-CUBE was launched into orbit on July 9th 2024. Among its hosted payloads is the Coral Dev Board Mini – a single-board computer developed by Google with an embedded Edge TPU designed for fast, low-power machine learning inference in constrained environments.

Alongside radiation shielding and thermal managements strategies, a technology demonstrator is being developed to assess the tolerance of these components to space radiation (e.g. protons, gamma rays). A central objective of the project is to determine not only how broadly COTS components can be applied in space systems, but also whether complex image processing tasks can be carried out onboard. This would reduce the need for data downlink and significantly improve the efficiency of Earth observation and communication missions.

Thesis goal

AITHER project launched a 10kg nanosatellite OOV-CUBE on 9th of July 2024. Besides other hosted payloads not relevant for this thesis it is carrying Coral Dev Board Mini developed by Google – a single-board computer with an embedded Edge TPU module primarily made to provide fast machine learning inferencing in a small form factor. Radiation shielding and thermal diffusion were already taken into consideration by design of the satellite. The software for AI inference has now to be developed and uploaded to the board in orbit. As a software a CNN for cloud segmentation was chosen. Even though efficient CNN architectures are already existing and were studied in scientific papers, the embedded implementation of a complex deep learning algorithm has its own caveats. Due to restricted resources provided by an embedded system such as limitations in flash and RAM-memory, CPU and TPU capacity and speed, there are certain decisions to be made and the tricks applied in order to implement the CNN model in Coral Dev Board mini flawlessly. This is the main goal of this thesis

While the satellite’s structure was designed with radiation shielding and thermal management in mind, the software for onboard inference still needs to be developed and uplinked in the orbit. This thesis focuses on the implementation of a convolutional neural network (CNN) for cloud segmentation from satellite imagery. Although lightweight CNN architectures already exist and have been described in recent scientific literature, porting such a model to an embedded platform introduces unique challenges. These include limitations in memory, computation power of both central processing unit (CPU) and tensor processing unit (TPU) as well as model format compatibility.

The goal of this thesis is to design, train and deploy a CNN model for cloud segmentation on the Coral Dev Board Mini. The core task is to adapt the model for efficient inference on the Edge TPU, overcoming hardware constraints while maintaining acceptable segmentation performance. This work serves as a demonstration of the potential to carry out deep learning inference onboard a satellite using COTS hardware.

**Chapter 1: Introduction**

Electronic components used in satellites and spacecraft are exposed to intense radiation, high-energy particles, and extreme temperature fluctuations. To ensure reliability in such harsh conditions, traditional space missions rely on certified space-grade hardware. However, this approach significantly increases development time and cost.

The AITHER project, launched by TU Berlin, aims to explore the feasibility of using commercial off-the-shelf (COTS) components in small satellites, particularly nanosatellites in the 10–100 kg range. As part of this initiative, the 10 kg nanosatellite OOV-CUBE was launched into orbit on July 9th, 2024. Among its hosted payloads is the Coral Dev Board Mini, a single-board computer equipped with an Edge TPU module designed for fast and efficient AI inference in embedded applications.

While the satellite’s hardware was designed with radiation shielding and thermal management in mind, the software for onboard inference still needs to be developed and uploaded. This thesis focuses on the implementation of a convolutional neural network (CNN) for cloud segmentation from satellite imagery. Although lightweight CNN architectures already exist and have been described in recent literature, porting such a model to an embedded platform introduces unique challenges. These include limitations in memory, computation, and model format compatibility.

The goal of this thesis is to design, train, and deploy a CNN model for cloud segmentation on the Coral Dev Board Mini. The core task is to adapt the model for efficient inference on the Edge TPU, overcoming hardware constraints while maintaining acceptable segmentation performance. This work serves as a demonstration of the potential to carry out deep learning inference onboard a satellite using COTS hardware.

The scope of the thesis is limited to software-side implementation and validation. No real-time satellite communication is covered, and radiation tolerance is considered only in context, not experimentally. The input dataset is restricted to the publicly available Landsat 8 satellite imagery.

To achieve this goal, the thesis first reviews the relevant CNN architectures for cloud detection, selects an appropriate model, and trains it using available satellite data. The trained model is then converted and quantized for deployment on the Edge TPU. Finally, inference performance is evaluated directly on the Coral Dev Board Mini.

The structure of the thesis is as follows:  
Chapter 2 provides background on cloud segmentation and embedded deep learning.  
Chapter 3 outlines the implementation steps, from model selection to deployment.  
Chapter 4 presents the evaluation results.  
Chapter 5 concludes the work and discusses possible future directions.