Earth and Space Science

Supporting Information for

DeepLandforms: A Deep Learning Computer Vision toolset applied to a prime use case for mapping planetary skylights

Giacomo Nodjoumi[1*], Riccardo Pozzobon[2,3], Francesco Sauro[4], Angelo Pio Rossi[1]

[1]Jacobs University Bremen gGmbH, Bremen, Germany
[2]INAF-Astronomical Observatory of Padova, Padova, Italy
[3]Department of Geosciences, University of Padova, Padova, Italy
[4]Department of Biological, Geological and Environmental Sciences, University of Bologna, Bologna, Italy

Contents of this file

Introduction

The following supporting information includes general insights on Deep Learning Object Detection and Instance Segmentation methodologies and their state-of-the-art (Text S1) including a brief overview of Facebook's Detectron2 (Text S2). In Text S3 is described the dataset preparation workflow while in Text S4 are described Image Masks with an example in Figure S1.

In Text S5 are listed the components of the *DeepLandforms* toolset that can be found in the repository with a more detailed description of the jupyter notebooks available in Text S6.

The Text S7 contains the Mask_rcnn_R_50_FPN model configuration used in this work.

Figures S2 and S3 contains additional examples of the comparison between brightness and DTM profiles for pits.

In Figure S4 are showed examples of multiple detections of the same landforms related to source images with different acquisition angle.

Figures from S5 to S8 show several examples of the results obtained by using the manual approach of Cushing et al., compared to the results obtained by using the tools based on the object detection (DeepLandforms-YOLOv5) and Instance Segmentation (DeepLandforms) methodologies.

Figure S9 to S11 shows an example of a training session monitored through tensorboard, in which are visible all the mAP for object detection and instance segmentation for each class type.

A summary of the main parameters essential to landforms description are available in Table S1.

Table S2 contains a list of acronyms found in the main document and here.

In Table S3 are resumed the ranges of the model training parameters while in Table S4 are summarized the average results obtained at the end of the model training, same as Figures S9 to S11.

The list of all the names of the source images used in this work and downloaded from the PDS Geosciences Node Orbital Data Explorer (ODE).

Text S1. State of the art in Deep Learning Object Detection and Segmentation methodologies

An Artificial Neural Network (ANN) can be intended as a complex computational model, inspired by biological neural networks, composed of "digital neurons" and used to easily solve complex problems. For Deep Learning (DL) we mean a Machine Learning (ML) technique that makes use of an artificial neural network based on a layered structure with different degrees of interconnection in which the first level is the data entry, the last is the level of results return and the intermediate levels can be defined as decision-making levels or a set of logical choices implemented autonomously by the artificial network (Hoeser & Kuenzer, 2020).

A Convolutional Neural Network (CNN) (Dhillon & Verma, 2020; Gu et al., 2018) is an algorithm that take in an input image, assigns importance (*weights*) to aspects/objects and learn how to differentiate among other aspects/objects and is composed mainly by Convolution Layers, Pooling Layers and Fully Connected Layers in different configurations.

There are several well-known and widely used CNN (Hoeser & Kuenzer, 2020), most used are R-CNN-based for object detection (Fast-R-CNN, Faster-R-CNN) (Girshick, 2015; Ren et al., 2016) and Instance Segmentation(Mask-R-CNN) (Chen et al., 2019; He et al., 2018, p.; Massa & Girshick, 2018), YOLO (Pham et al., 2020), ResNet (Targ et al., 2016), MobileSSD (Howard et al., 2017). and are particularly used in Computer Vision (CV) with applications for the classification and recognition of objects present. in an image or video, both static and dynamic, for example the recognition of license plates, faces, object tracking, etc.

When training new models, it is possible to train them from scratch only if a custom training dataset is big enough, otherwise an underfitting problem will occur.

A model is considered overfitted when it has excellent performance on trained data but poor performance on newer data while is considered underfitted when it has global poor performances related to a very small dataset (Amazon Machine Learning, 2021; Zhang et al., 2021).

Almost all state-of-the-art public Earth-related datasets such as MS COCO, ImageNet (Deng et al., 2009; Lin et al., 2015), contain up to several millions of labels and associated images and are commonly used for training above-mentioned architectures, granting a very good generalization of the problem of detection and segmentation without underfitting or overfitting. Models trained with such datasets are used to compute benchmarks for those architectures and evaluate their performance. Those benchmarks are further used as reference for newer and modified architectures (Wu et al., 2019b).

A possible solution to overfitting and underfitting is a method named Transfer Learning (Neyshabur et al., 2020, 2021; Tan et al., 2018; Weiss et al., 2016) in which a model pre-trained on a very large dataset, such as the above-mentioned ones, is partially re-trained to adapt it to a custom dataset. For instance, an object detection model pre-trained on a large dataset containing thousands of trees images, can be re-trained on a custom smaller dataset containing plants.

An oversimplification of the training phase of a model consists of several cycles (*epochs*) in which data (*batch size*) are fed to the network, weights are computed at each layer, an output is produced, and an estimation of the error (*loss*) is computed. Then the difference between the obtained result and the correct one is backpropagated to each layer in order to adjust the corresponding weights. This process is regulated by a parameter called *learning rate*, a value between 0 and 1 that is multiplied by the loss gradient rate e computed difference in order to avoid over adaptation to that specific data, losing effectiveness to other data (Google Developers, 2021)

The common evaluation metric used in object detection and image segmentation based on COCO format is the mean Average Precision (mAP). This parameter can be described as the area under the precision-recall curve. Precision is the parameter that corresponds to the ratio of True Positive (TP) predictions and the sum of TP and False Positive (FP) predictions. Recall is the ratio of TP and the sum of TP and False Negatives (FN). The method to decide whether a prediction is positive, or negative is based on the Intersection over Union (IoU). IoU is the ratio between the intersection between the predicted shape area and the ground truth shape area divided by the sum of the area of those shapes. The Average Precision score is computed

automatically for each class at different IoU thresholds, commonly at 0.5, 0.75 and 0.95 and then averaged to obtain the global mAP and for each class (Ghorbanzadeh et al., 2019; MAY 6 & Read, 2020).

Text S2. Facebook AI Research's Detectron2

Detectron2 is a library developed by the Facebook AI Research team, and is an evolution of Detectron library (Wu et al., 2019a, 2020) and maskrcnn-benchmark (Massa & Girshick, 2018) Detectron2's developers provide several models such as Faster- and Mask- Region-based Convolutional Neural Network (Faster-R-CNN, Mask R-CNN), RetinaNet, DensePose, Cascade R-CNN, Panoptic FPN, and TensorMask, including their relative baseline pre-trained versions (Wu et al., 2019b) which can be used to further train custom models.

All of these are state-of-the-art trained architectures specialized in object detection, image segmentation, instance segmentation and pose estimation. Moreover, this library is well documented (Facebook AI Research, 2020) and the code is maintained periodically, including features updates and integrations of newer models (Wu et al., 2019b). Considering all these features, Detectron2 has been chosen as a core library to develop new tools for planetary mapper. Pytorch has been used as a framework (Paszke et al., 2019).

All included pre-trained models are compatible with the tool presented in this work, but for the specific task of instance segmentation for mapping landforms, the Mask R-CNN networks have been considered the defaults since are specific for object instance segmentation tasks (He et al., 2018).

Text S3. Deep Learning main workflow steps

Depending on the data size and format, the labelling software and the DeepLearning architecture and approach, may vary slightly. Common data pre-processing includes a data format conversion from the source data format to a lighter format compatible with both the labelling tool and the Deep Learning tool.

Common file formats include jpeg, png and tiff, while the file size is usually lower than 1024 x 1024 pixel.

To meet those requirements, images can be resized, loosing spatial resolution, cropped, loosing portion of the image and tiled, increasing the number of the images.

For instance, in this work have been used Reduced Data Record (RDR) RED channel data acquired by the High-Resolution Imaging Science Experiment (HiRISE) on board the Mars Reconnaissance Orbiter (MRO) and consist of calibrated and map-projected images at highest spatial resolution possible, provided in JPEG2000 (JP2) file format. File size of these images may reach few Gigabytes with used

Data are then ingested in labeling tool that produce ancillary data containing all the labels in a format compatible with the Deep Learning training pipeline. For Object Detection, labels consist of bounding boxes containing a single object of interest, while for Instance Segmentation, labels consist of polygonal shape containing a single object of interest. For both types of labels, a single class category must be assigned.

Before of within the training pipeline, the source dataset is divided into three main sub-datasets: train, test and valid, with a common rule-of-thumb proportion of 70:20:10%, meaning that 70% of the data will be used for train dataset, 20% for the test dataset and 10% for validating the trained model.

Train and test dataset are then used in the training pipeline, in which the test dataset is used for evaluating the model while training.

After the completion of the training, the model is evaluated using the valid dataset, to assess how it performs with unseen data.

Text S4. Image masks

The filters that are used to create masks can be based on both users defined custom values or no-data values embedded in image's metadata. For instance, some images may have Not-a-Number (NaN) values as no-data, while others may have other fixed values that are commonly provided in dataset ancillary files if not embedded in the data.

Segmentation results of Mask R-CNNs have instance masks as results that can cover the whole image (image segmentation) or only identified objects (instance segmentation) as shown in Figure S1.

Text S5. Docker-compose and dockerfiles

The major advantage of usage of docker platform resides in the capabilities of running services in an instantiated environment, independent from the host operating system without worrying of library dependencies, compilers, interpreters and so on, thus providing great cross-platform compatibility. Moreover, docker containers can be shared both as pre-built container images for fast deployment and as a docker building recipe named dockerfile. Those recipes can be customized by combining with other dockerfiles and, or, with pre-build docker images. Dockerfiles can also be used in combination with docker-compose (Merkel, 2014), a tool for running multi-container Docker applications. DeepLandforms repository (Nodjoumi, 2021) contains both single separate dockerfiles and a docker-compose configuration files for automatic building of the necessary docker images with only minor configuration requirements by the user. These files are available both in Github repository (Nodjoumi, 2021) and Zenodo (Nodjoumi, 2021).

The repository is structured as follows:

- Dockerfiles for creating the main tool docker images containing training and inference jupyter notebooks,
- Dockerfile for creating tensorboard docker image, an open-source tool for monitoring model training [81],
- Dockerfile for creating labelme docker image, an open-source tool for image polygonal annotation in Common Object Context (COCO) label format (Lin et al., 2015),
- Dockerfile for creating ImageProcessingUtils docker image, an open-source tool for image resizing and tiles,
- A folder containing all the developed notebooks and addons,

- README, containing brief description and initial guidelines.

Text S6. DeepLandforms Notebooks

DeepLandforms-Training notebook is an implementation of the Detectron2 Library's training components in a jupyter notebook, in which is possible to control all the main hyperparameters mentioned in Text S1, such as Epochs, LearningRate, batch size and more, and other parameters such the percentage used to create the train, test, and valid datasets.

The notebook automatically creates such datasets and visualize pie-charts representing each dataset with class distributions.

After dataset creation the notebook create an ancillary file containing all the classes that has been trained and save it in the same directory of the trained model.

DeepLandforms-Segmentation notebook is an implementation of the Detectron2 Library's inference components. This notebook includes several custom functions which are used to compute the world coordinates of the detection masks and their conversion into vectorial data, including geopackage creation. Moreover, a specific function has been developed to convert the detection masks into label files in COCO json format that can be directly used to perform new training sessions.

Text S7. Mask_rcnn_R_50_FPN model configuration

This model configuration can be visualized by using the following lines of code after loading properly the corresponding configuration.

Python3 code:

```
from detectron2.engine import DefaultTrainer
trainer = DefaultTrainer(cfg)
trainer.resume_or_load(resume=False)
```

```
GeneralizedRCNN(
  (backbone): FPN(
    (fpn_lateral2): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (fpn_lateral3): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (fpn_lateral4): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output4): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (fpn_lateral5): Conv2d(2048, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output5): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (top_block): LastLevelMaxPool()
    (bottom_up): ResNet(
      (stem): BasicStem(
        (conv1): Conv2d(
```

```
    3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False
    (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
   )
  )
  (res2): Sequential(
   (0): BottleneckBlock(
    (shortcut): Conv2d(
     64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
     (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv1): Conv2d(
     64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False
     (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
    )
    (conv2): Conv2d(
     64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
     (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
    )
    (conv3): Conv2d(
     64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
     (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
   )
   (1): BottleneckBlock(
    (conv1): Conv2d(
     256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False
     (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
    )
    (conv2): Conv2d(
     64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
     (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
    )
    (conv3): Conv2d(
     64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
     (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
   )
   (2): BottleneckBlock(
    (conv1): Conv2d(
     256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False
     (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
    )
    (conv2): Conv2d(
     64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
     (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
    )
    (conv3): Conv2d(
```

```
      64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
  )
)
(res3): Sequential(
  (0): BottleneckBlock(
    (shortcut): Conv2d(
      256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False
      (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
    )
    (conv1): Conv2d(
      256, 128, kernel_size=(1, 1), stride=(2, 2), bias=False
      (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
    )
    (conv2): Conv2d(
      128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
    )
    (conv3): Conv2d(
      128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
    )
  )
  (1): BottleneckBlock(
    (conv1): Conv2d(
      512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
    )
    (conv2): Conv2d(
      128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
    )
    (conv3): Conv2d(
      128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
    )
  )
  (2): BottleneckBlock(
    (conv1): Conv2d(
      512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
    )
    (conv2): Conv2d(
      128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
    )
```

```
    (conv3): Conv2d(
      128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
    )
  )
  (3): BottleneckBlock(
    (conv1): Conv2d(
      512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
    )
    (conv2): Conv2d(
      128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
    )
    (conv3): Conv2d(
      128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
    )
  )
)
(res4): Sequential(
  (0): BottleneckBlock(
    (shortcut): Conv2d(
      512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=False
      (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
    )
    (conv1): Conv2d(
      512, 256, kernel_size=(1, 1), stride=(2, 2), bias=False
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv2): Conv2d(
      256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv3): Conv2d(
      256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
    )
  )
  (1): BottleneckBlock(
    (conv1): Conv2d(
      1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv2): Conv2d(
      256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
```

)
(conv3): Conv2d(
256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
(norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
)
)
(2): BottleneckBlock(
(conv1): Conv2d(
1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
(norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
)
(conv2): Conv2d(
256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
(norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
)
(conv3): Conv2d(
256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
(norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
)
)
(3): BottleneckBlock(
(conv1): Conv2d(
1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
(norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
)
(conv2): Conv2d(
256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
(norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
)
(conv3): Conv2d(
256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
(norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
)
)
(4): BottleneckBlock(
(conv1): Conv2d(
1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
(norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
)
(conv2): Conv2d(
256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
(norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
)
(conv3): Conv2d(
256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
(norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
)

```
  )
  (5): BottleneckBlock(
    (conv1): Conv2d(
      1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv2): Conv2d(
      256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv3): Conv2d(
      256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
    )
  )
)
(res5): Sequential(
  (0): BottleneckBlock(
    (shortcut): Conv2d(
      1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False
      (norm): FrozenBatchNorm2d(num_features=2048, eps=1e-05)
    )
    (conv1): Conv2d(
      1024, 512, kernel_size=(1, 1), stride=(2, 2), bias=False
      (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
    )
    (conv2): Conv2d(
      512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
    )
    (conv3): Conv2d(
      512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=2048, eps=1e-05)
    )
  )
  (1): BottleneckBlock(
    (conv1): Conv2d(
      2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
    )
    (conv2): Conv2d(
      512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
    )
    (conv3): Conv2d(
      512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=2048, eps=1e-05)
```

```
      )
     )
     (2): BottleneckBlock(
      (conv1): Conv2d(
       2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
       (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
      )
      (conv2): Conv2d(
       512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
       (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
      )
      (conv3): Conv2d(
       512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False
       (norm): FrozenBatchNorm2d(num_features=2048, eps=1e-05)
      )
     )
    )
   )
  )
 (proposal_generator): RPN(
  (rpn_head): StandardRPNHead(
   (conv): Conv2d(
    256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)
    (activation): ReLU()
   )
   (objectness_logits): Conv2d(256, 3, kernel_size=(1, 1), stride=(1, 1))
   (anchor_deltas): Conv2d(256, 12, kernel_size=(1, 1), stride=(1, 1))
  )
  (anchor_generator): DefaultAnchorGenerator(
   (cell_anchors): BufferList()
  )
 )
 (roi_heads): StandardROIHeads(
  (box_pooler): ROIPooler(
   (level_poolers): ModuleList(
    (0): ROIAlign(output_size=(7, 7), spatial_scale=0.25, sampling_ratio=0, aligned=True)
    (1): ROIAlign(output_size=(7, 7), spatial_scale=0.125, sampling_ratio=0, aligned=True)
    (2): ROIAlign(output_size=(7, 7), spatial_scale=0.0625, sampling_ratio=0, aligned=True)
    (3): ROIAlign(output_size=(7, 7), spatial_scale=0.03125, sampling_ratio=0, aligned=True)
   )
  )
  (box_head): FastRCNNConvFCHead(
   (flatten): Flatten(start_dim=1, end_dim=-1)
   (fc1): Linear(in_features=12544, out_features=1024, bias=True)
   (fc_relu1): ReLU()
   (fc2): Linear(in_features=1024, out_features=1024, bias=True)
   (fc_relu2): ReLU()
```

```
    )
    (box_predictor): FastRCNNOutputLayers(
      (cls_score): Linear(in_features=1024, out_features=8, bias=True)
      (bbox_pred): Linear(in_features=1024, out_features=28, bias=True)
    )
    (mask_pooler): ROIPooler(
      (level_poolers): ModuleList(
        (0): ROIAlign(output_size=(14, 14), spatial_scale=0.25, sampling_ratio=0, aligned=True)
        (1): ROIAlign(output_size=(14, 14), spatial_scale=0.125, sampling_ratio=0, aligned=True)
        (2): ROIAlign(output_size=(14, 14), spatial_scale=0.0625, sampling_ratio=0, aligned=True)
        (3): ROIAlign(output_size=(14, 14), spatial_scale=0.03125, sampling_ratio=0, aligned=True)
      )
    )
    (mask_head): MaskRCNNConvUpsampleHead(
      (mask_fcn1): Conv2d(
        256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)
        (activation): ReLU()
      )
      (mask_fcn2): Conv2d(
        256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)
        (activation): ReLU()
      )
      (mask_fcn3): Conv2d(
        256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)
        (activation): ReLU()
      )
      (mask_fcn4): Conv2d(
        256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)
        (activation): ReLU()
      )
      (deconv): ConvTranspose2d(256, 256, kernel_size=(2, 2), stride=(2, 2))
      (deconv_relu): ReLU()
      (predictor): Conv2d(256, 7, kernel_size=(1, 1), stride=(1, 1))
    )
  )
)
```

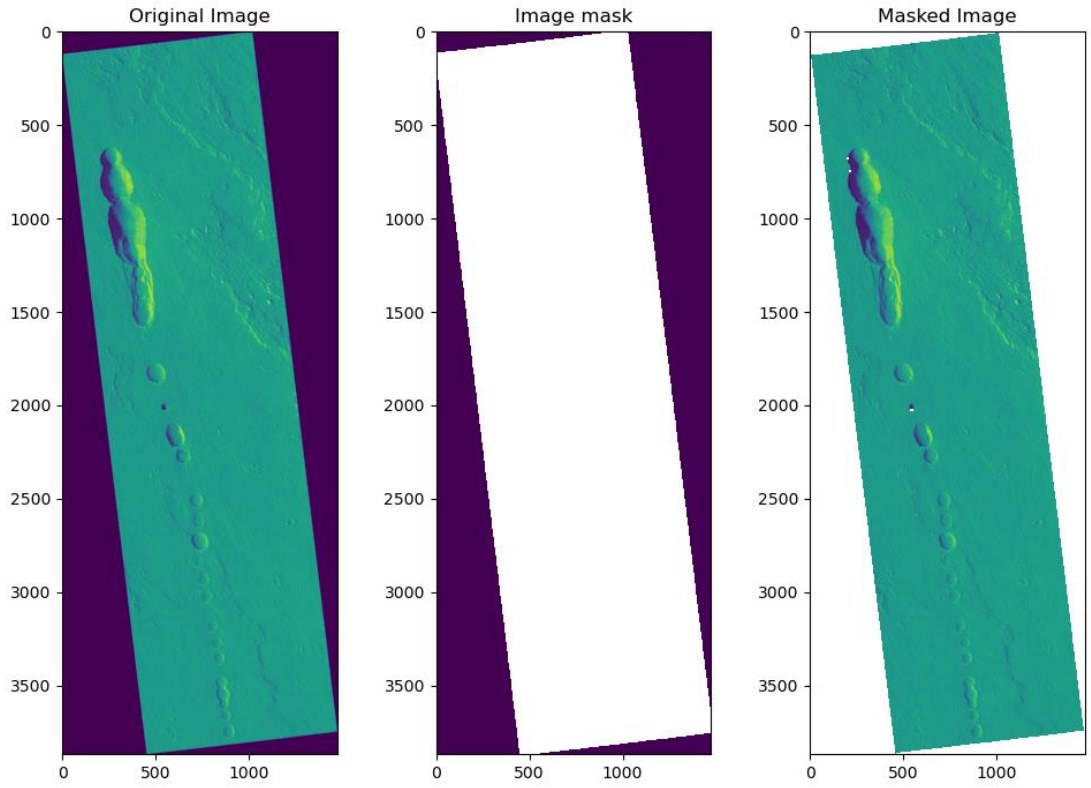


Figure S1. Example of masks on HiRISE image. Purple color represents nodata value.

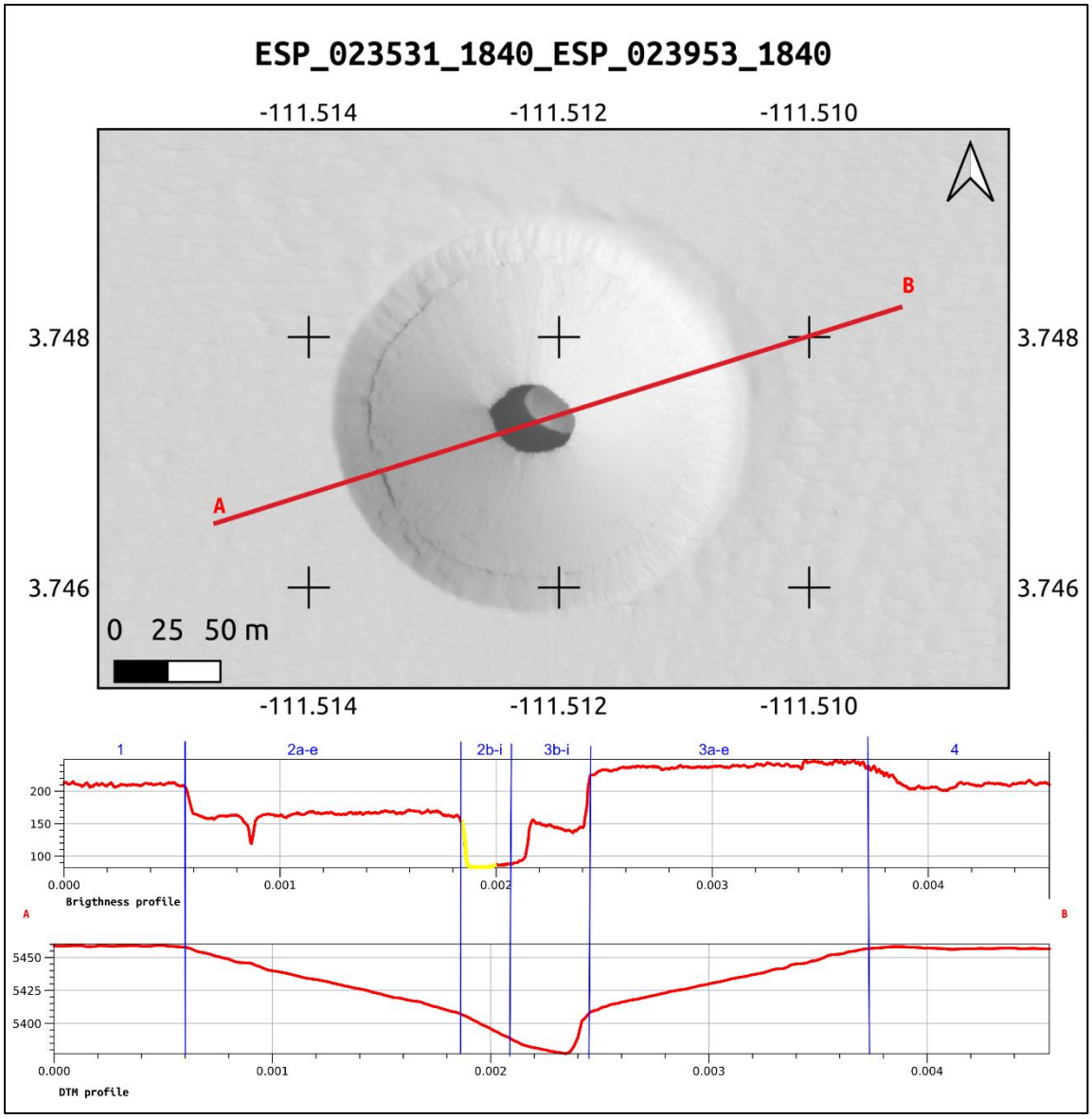


Figure S2 Comparison of brightness and DTM profiles for a Type-1 nested in Type-2 on ESP_23531-ESP_23953 stereo pair, red channel. In blue, an attempt to identify the correspondence of the brightness sections on a DTM derived from photogrammetry of HiRISE stereo pair. In yellow, the portion of the DTM profile interpolated by the Socet Set ((c) BAE Systems) software. This is due to lack of usable data in both the images processed using photogrammetry **(*HiRISE | About HiRISE Digital Terrain Models*, 2021)**

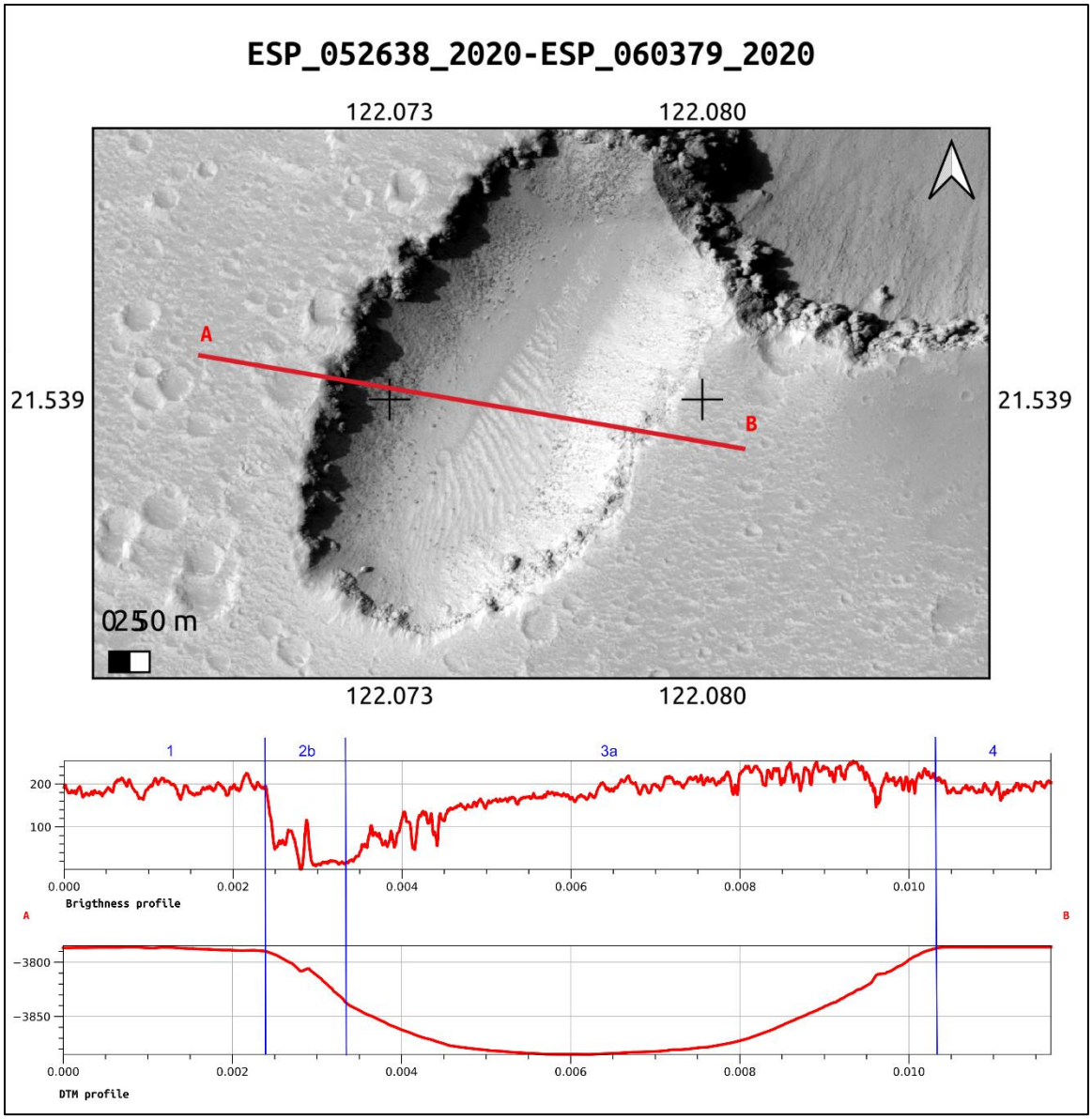
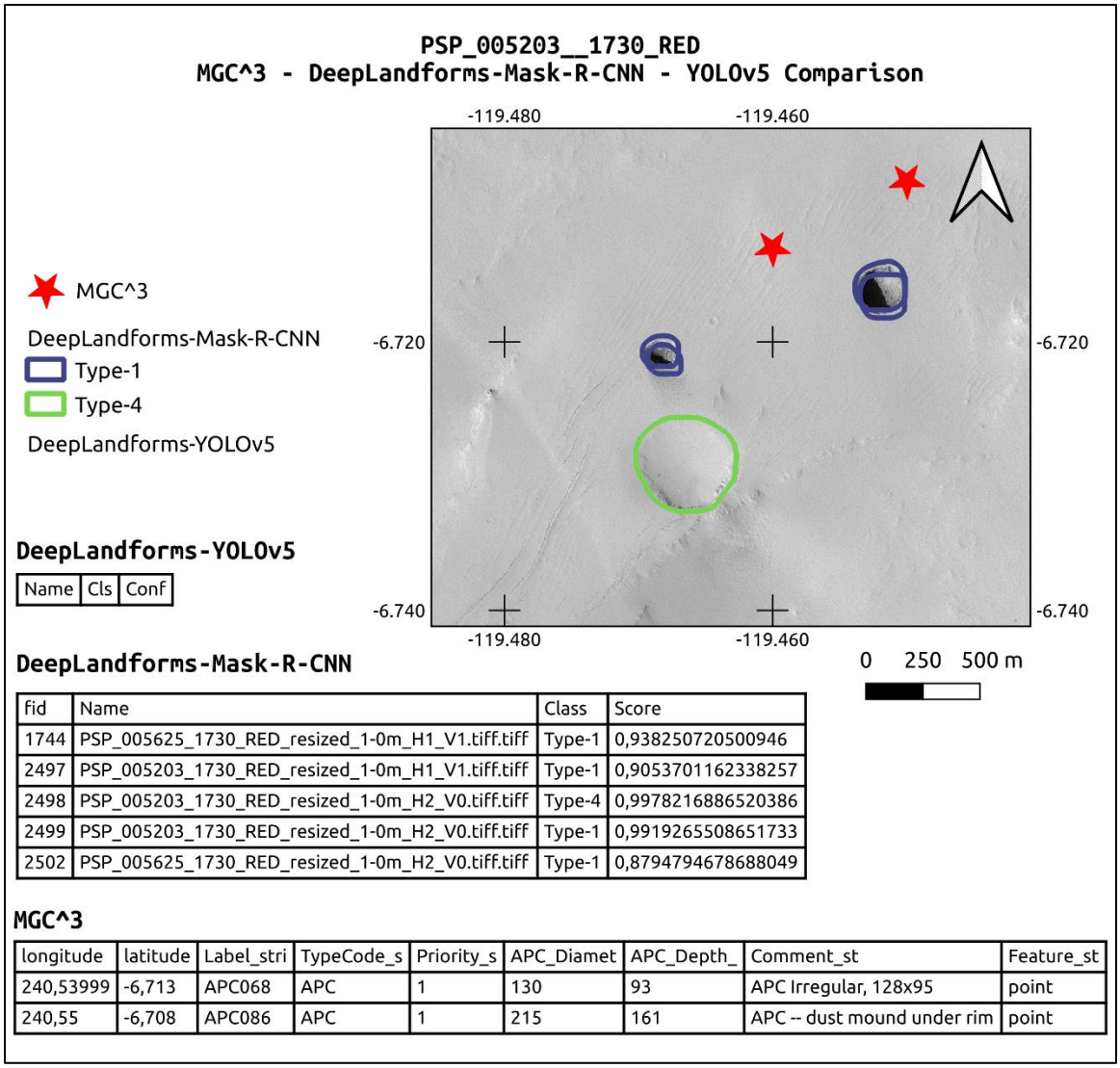


Figure S3. Comparison of brightness and DTM profiles for a Type-3 on ESP_23531_RED ortho image, the DTM is derived from ESP_23531-ESP_23953 stereo pair, red channel. In blue, an attempt to identify the correspondence of the brightness sections on a DTM derived from photogrammetry of HiRISE stereo pair. In yellow, the portion of the DTM profile interpolated by the Socet Set ((c) BAE Systems) software. This is due to lack of usable data in both the images processed using photogrammetry **(*HiRISE | About HiRISE Digital Terrain Models*, 2021)**



DeepLandforms-YOLOv5

Name	Cls	Conf

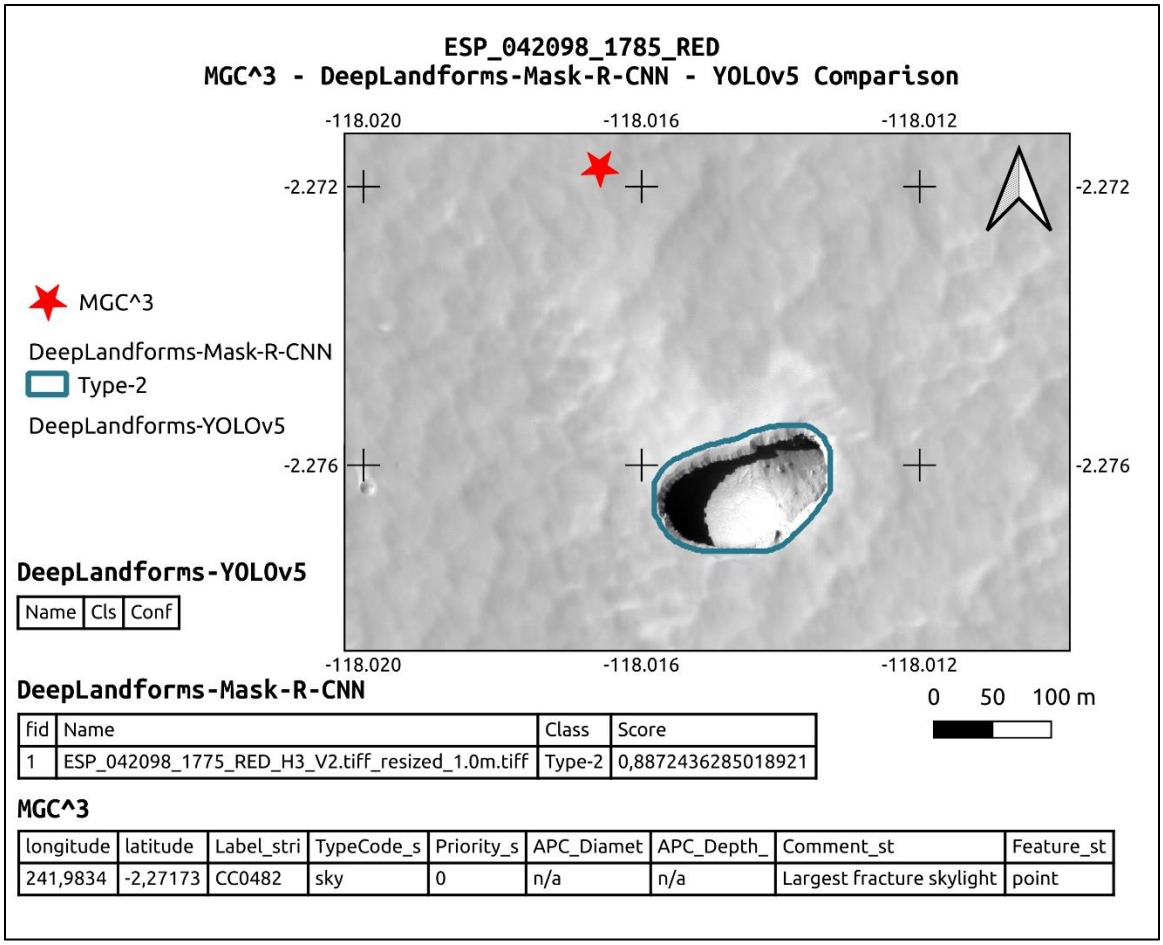
DeepLandforms-Mask-R-CNN

fid	Name	Class	Score
1744	PSP_005625_1730_RED_resized_1-0m_H1_V1.tiff.tiff	Type-1	0,938250720500946
2497	PSP_005203_1730_RED_resized_1-0m_H1_V1.tiff.tiff	Type-1	0,9053701162338257
2498	PSP_005203_1730_RED_resized_1-0m_H2_V0.tiff.tiff	Type-4	0,9978216886520386
2499	PSP_005203_1730_RED_resized_1-0m_H2_V0.tiff.tiff	Type-1	0,9919265508651733
2502	PSP_005625_1730_RED_resized_1-0m_H2_V0.tiff.tiff	Type-1	0,8794794678688049

MGC^3

longitude	latitude	Label_stri	TypeCode_s	Priority_s	APC_Diamet	APC_Depth_	Comment_st	Feature_st
240,53999	-6,713	APC068	APC	1	130	93	APC Irregular, 128x95	point
240,55	-6,708	APC086	APC	1	215	161	APC -- dust mound under rim	point

Figure S4. Example of multiple polygons, slightly mis-aligned, detected on multiple images acquired of the same area compared to entries of the MGC^3 database with both attribute tables displayed. The results, stored in a geopackage, were opened in a GIS environment where can be styled using different colors for each class. The offset of MGC^3 points is probably caused by the coordinate conversion of the source MGC^3 csv file into shapefile.



DeepLandforms-YOLOv5

Name	Cls	Conf

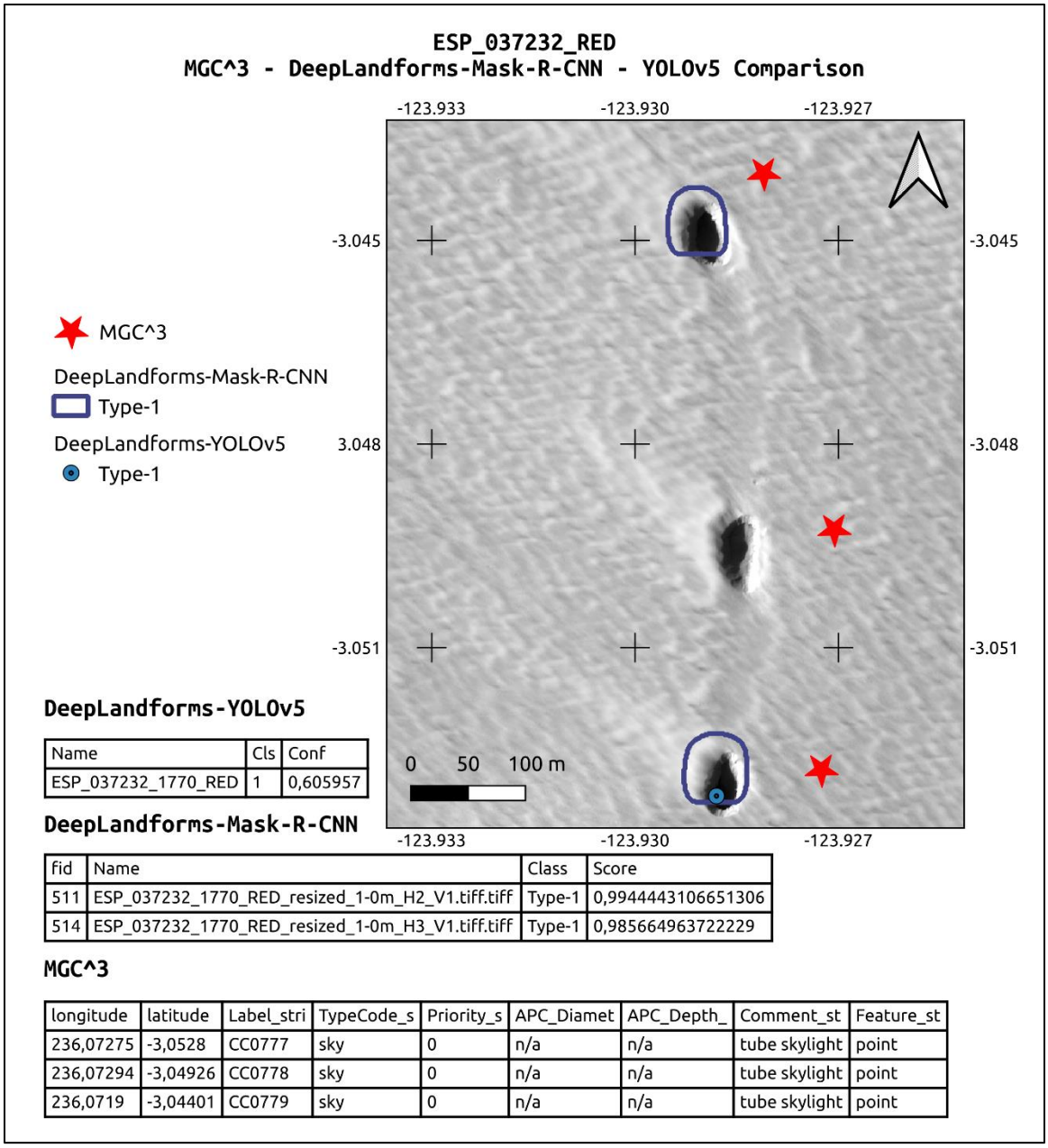
DeepLandforms-Mask-R-CNN

fid	Name	Class	Score
1	ESP_042098_1775_RED_H3_V2.tiff_resized_1.0m.tiff	Type-2	0,8872436285018921

MGC^3

longitude	latitude	Label_stri	TypeCode_s	Priority_s	APC_Diamet	APC_Depth_	Comment_st	Feature_st
241,9834	-2,27173	CC0482	sky	0	n/a	n/a	Largest fracture skylight	point

Figure S5. Example of detection on HiRISE Red channel image and comparison between MGC^3, DeepLandforms-YOLOv5 object detection and DeepLandforms-Mask-R-CNN instance segmentation. In this case Mask-R-CNN has difficulties to classify since the landform has properties similar to both Type-2, Type-1 and Type-3. Tables show attributes of fields in the shapefiles. The offset of MGC^3 points is probably caused by the coordinate conversion of the source MGC^3 csv file into shapefile.



DeepLandforms-YOLOv5

Name	Cls	Conf
ESP_037232_1770_RED	1	0,605957

DeepLandforms-Mask-R-CNN

fid	Name	Class	Score
511	ESP_037232_1770_RED_resized_1-0m_H2_V1.tiff.tiff	Type-1	0,9944443106651306
514	ESP_037232_1770_RED_resized_1-0m_H3_V1.tiff.tiff	Type-1	0,985664963722229

MGC^3

longitude	latitude	Label_stri	TypeCode_s	Priority_s	APC_Diamet	APC_Depth_	Comment_st	Feature_st
236,07275	-3,0528	CC0777	sky	0	n/a	n/a	tube skylight	point
236,07294	-3,04926	CC0778	sky	0	n/a	n/a	tube skylight	point
236,0719	-3,04401	CC0779	sky	0	n/a	n/a	tube skylight	point

Figure S6. Example of detection on HiRISE Red channel image and comparison between MGC^3, DeepLandforms-YOLOv5 object detection and DeepLandforms-Mask-R-CNN instance segmentation. Tables show attributes of fields in the shapefiles. The offset of MGC^3 points is probably caused by the coordinate conversion of the source MGC^3 csv file into shapefile.

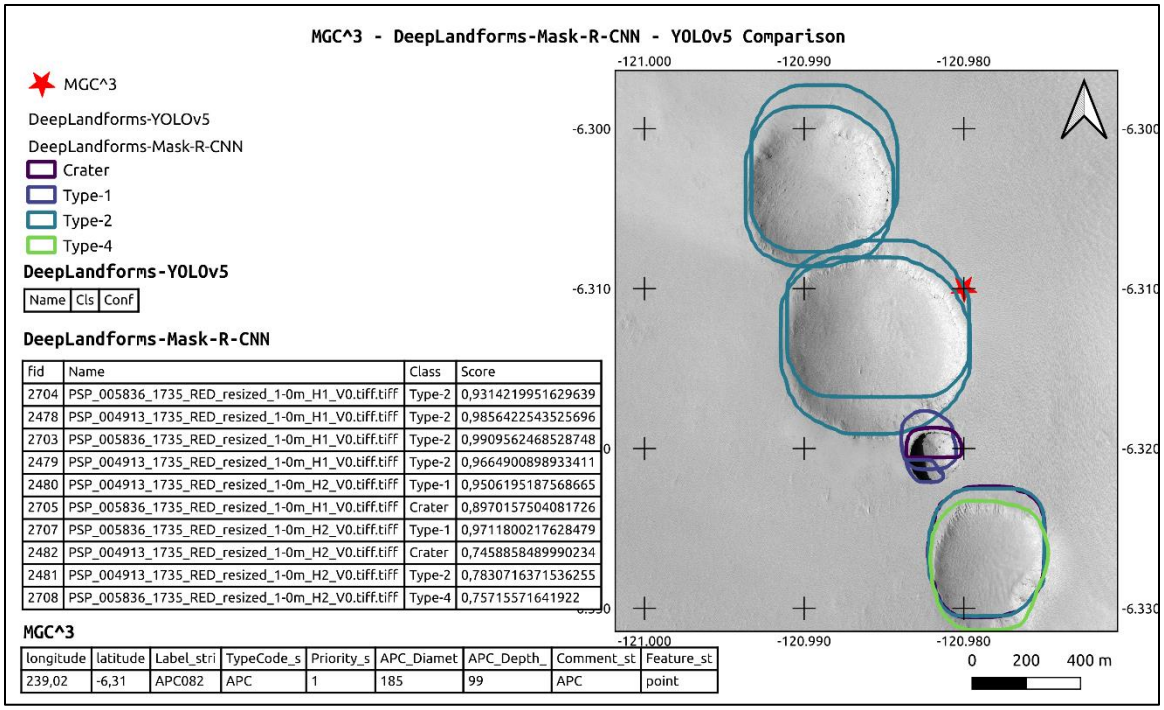
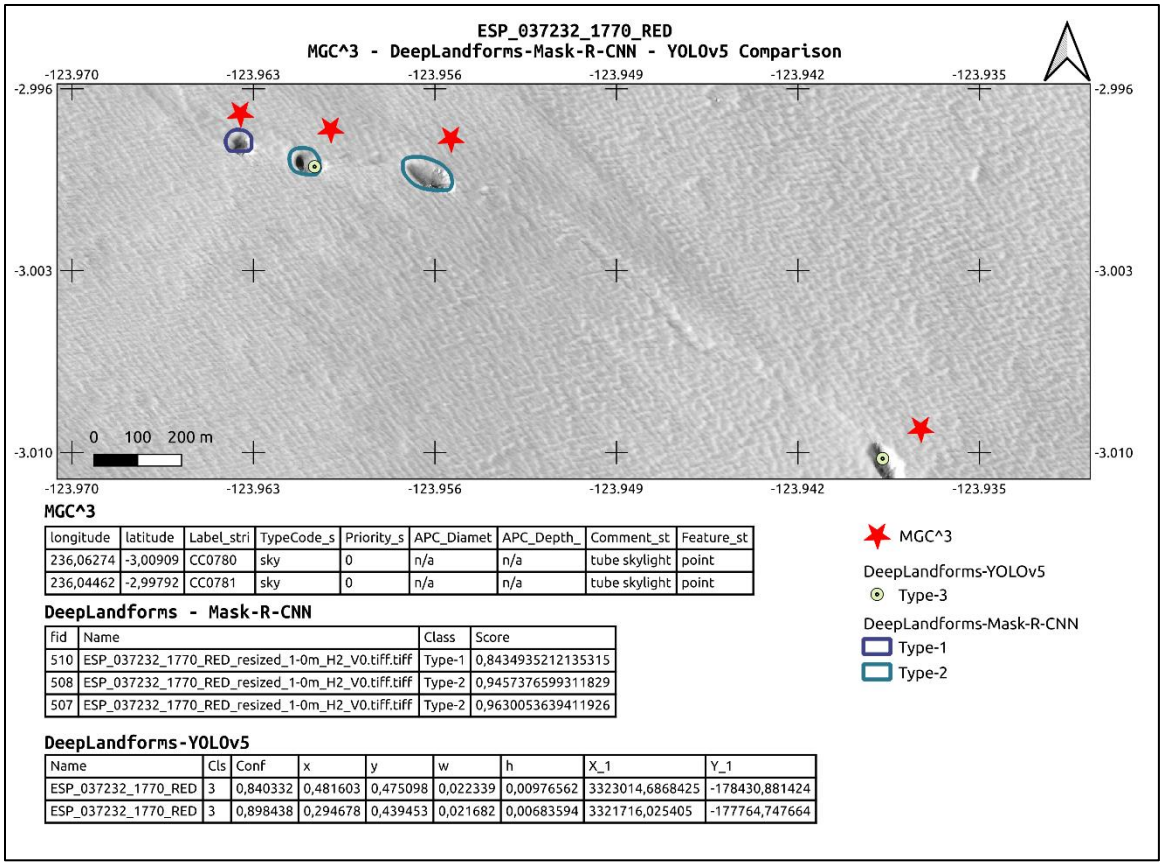


Figure S7. Example of detection on HiRISE Red channel image and comparison between MGC^3, DeepLandforms-YOLOv5 object detection and DeepLandforms-Mask-R-CNN instance segmentation. Multiple misclassification and segmentation error occurs. Classification errors, such as Type-1/Crater are mostly caused by the small training dataset. Segmentation errors are caused by both the small dataset and by inference performed on tiles. For instance, Type-1 and Type-2 have straight horizontal lines that correspond to the edge of the tile. This specific error can be mitigated by using larger tiles or by implementing a sliding-window analysis. Tables show attributes of fields in the shapefiles. The offset of MGC^3 points is probably caused by the coordinate conversion of the source MGC^3 csv file into shapefile.



MGC^3

longitude	latitude	Label_stri	TypeCode_s	Priority_s	APC_Diamet	APC_Depth_	Comment_st	Feature_st
236,06274	-3,00909	CC0780	sky	0	n/a	n/a	tube skylight	point
236,04462	-2,99792	CC0781	sky	0	n/a	n/a	tube skylight	point

DeepLandforms - Mask-R-CNN

fid	Name	Class	Score
510	ESP_037232_1770_RED_resized_1-0m_H2_V0.tiff.tiff	Type-1	0,8434935212135315
508	ESP_037232_1770_RED_resized_1-0m_H2_V0.tiff.tiff	Type-2	0,9457376599311829
507	ESP_037232_1770_RED_resized_1-0m_H2_V0.tiff.tiff	Type-2	0,9630053639411926

DeepLandforms-YOLOv5

Name	Cls	Conf	x	y	w	h	X_1	Y_1
ESP_037232_1770_RED	3	0,840332	0,481603	0,475098	0,022339	0,00976562	3323014,6868425	-178430,881424
ESP_037232_1770_RED	3	0,898438	0,294678	0,439453	0,021682	0,00683594	3321716,025405	-177764,747664

Figure S8. Example of detections on full HiRISE Red channel image. The first two shapes on top, present segmentation, and classification errors due to the inference that was performed on tiles instead of the full image. Other shapes present misclassification errors mostly due to the very small training datasets. Tables show attributes of fields in the shapefiles. These landforms are not present in the MGC^3.

Figure S9. Mean Average Precision values for the object detection, shows an overall average value compared to the Detectron2's benchmarks (Wu et al., 2019b) except for Type-4 and Crater landforms.

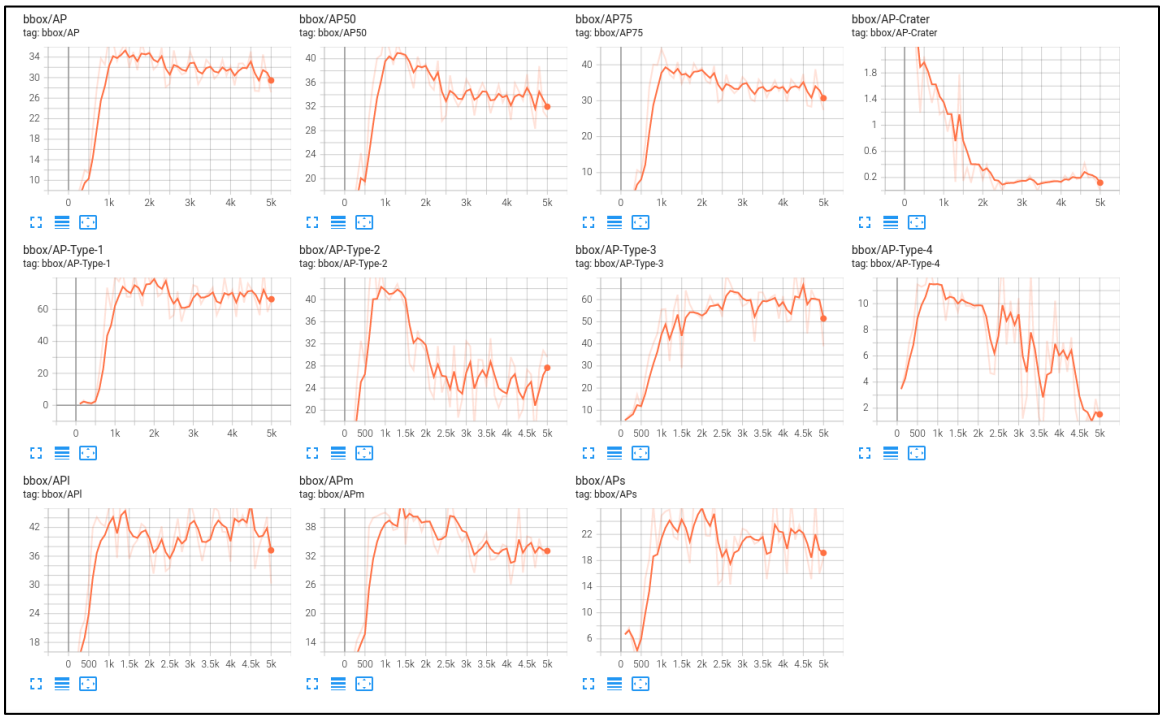


Figure S10. Mean Average Precision values for the segmentation, shows an overall average value compared to the Detectron2's benchmarks (Wu et al., 2019b) except for Type-4 and Crater landforms.

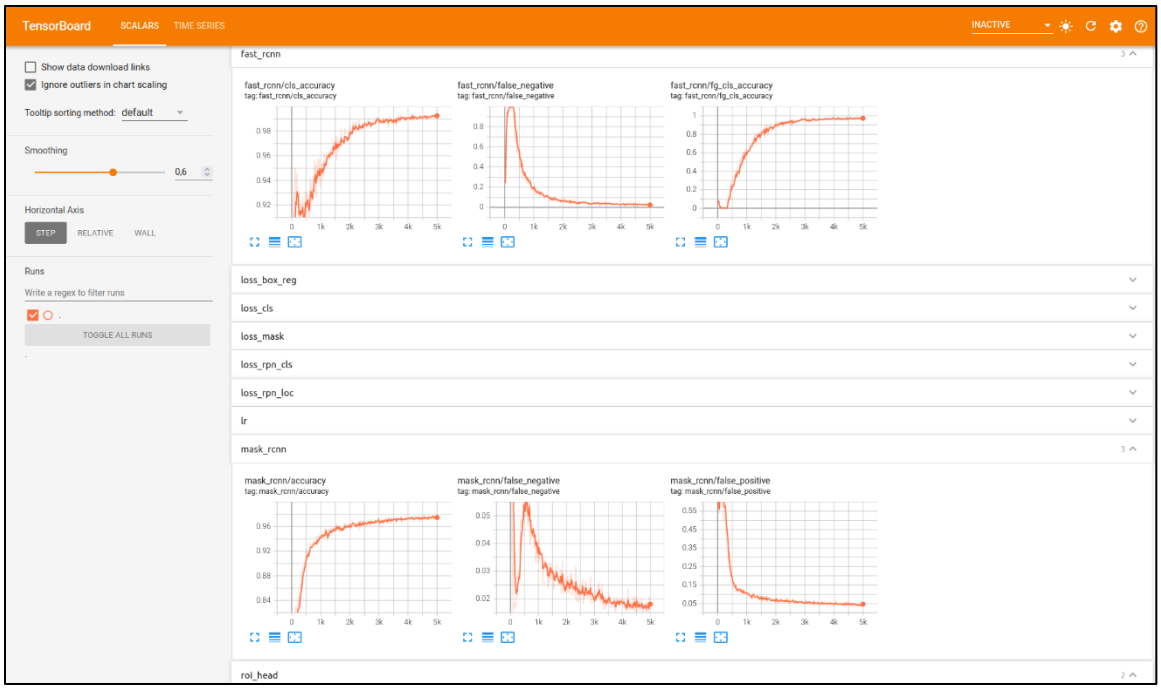


Figure S11. Overall training results for both object detection and instance segmentation visualized through *tensorboard* interface.

Parameter	**Description**
Shape	
Size	Width, Length
Height	Height
Texture	The frequency of tonal changes
Pattern	The spatial arrangement of objects or elements in a given area.
Tone/Hue	Black&White brightness and color hue, expression of the amount and type of light reflected (or emitted) by an object and reflection of the environmental condition during the acquisition
Location/Association	Spatial relations between an object and its surroundings

Table S1. Main parameters that describe a landform. **(Tempfli et al., 2009)**.

Acronym	Meaning
AI	Artificial Intelligence
Cascade-R-CNN	Cascade Regional CNN
CNN	Convolutional Neural Network
COCO	Common Object Context
CTX	Context Camera
CUDA	Compute Unified Device Architecture
DEM	Digital Elevation Model
DL	Deep Learning
DN	Digital Numbers
DTM	Digital Terrain Model
EDR	Experiment Data Record
FN	False Negative
FP	False Positive
Faster-R-CNN	Faster Regional CNN
GDAL	Geospatial Data Abstraction Software Library
GIS	Geographic Information System
GPU	Graphical Processor Unit
HiRISE	High Resolution Imaging Science Experiment
ImS	Image Segmentation
InS	Instance Segmentation
LRO	Lunar Reconnaissance Orbiter
Mask-R-CNN	Mask Regional CNN
mAP	Mean Average Precision
ML	Machine Learning
MRO	Mars Reconnaissance Orbiter
NAC	Narrow Angle Camera
NIR	Near InfraRed
OD	Object Detection
ODE	Orbital Data Explorer
OGC	Open Geospatial Consortium
Panoptic FPN	Panoptic Feature Proposal Network
RAM	Random Access Memory
R-CNN	Regional CNN
RDR	Reduced Data Record
RetinaNet	RetinaNetwork
TP	True Positive
VIS	Visible
VRAM	Video RAM
WAC	Wide Angle Camera
YOLO	You Only Look Once

Table S2. List of main acronyms present in the main paper.

Parameter	Value
Epochs	1000 to 5000
Batch size	4 to 8
LearningRate	0,00025 to 0,002

Table S3. Ranges of parameters used in all training tests.

Type	Detection mAP	Segmentation mAP
Global	32	30
Type-1	65	55
Type-2	29	23
Type-3	38	25
Type-4	1	1
Crater	1	1

Table S4. Mean mAP obtained at the end of the training.

HiRISE RDR ODE-PDS Filename
ESP_012600_1655_RED.JP2
ESP_013167_1785_RED.JP2
ESP_013589_1785_RED.JP2
ESP_013681_1765_RED.JP2
ESP_013879_1720_RED.JP2
ESP_014077_1660_RED.JP2
ESP_014380_1775_RED.JP2
ESP_016213_1720_RED.JP2
ESP_016305_2265_RED.JP2
ESP_016411_1605_RED.JP2
ESP_016767_1785_RED.JP2
ESP_016978_1730_RED.JP2
ESP_017057_1715_RED.JP2
ESP_017202_1685_RED.JP2
ESP_019259_1715_RED.JP2
ESP_019272_1980_RED.JP2
ESP_019351_1795_RED.JP2
ESP_019852_1810_RED.JP2
ESP_019957_2220_RED.JP2

ESP_019984_1795_RED.JP2
ESP_019997_1975_RED.JP2
ESP_020246_2185_RED.JP2
ESP_021738_1625_RED.JP2
ESP_022661_1705_RED.JP2
ESP_023531_1840_RED.JP2
ESP_024481_1605_RED.JP2
ESP_024929_1720_RED.JP2
ESP_025892_1780_RED.JP2
ESP_026695_2020_RED.JP2
ESP_026907_1705_RED.JP2
ESP_028370_2215_RED.JP2
ESP_028450_1730_RED.JP2
ESP_028700_2085_RED.JP2
ESP_028793_1655_RED.JP2
ESP_035254_1585_RED.JP2
ESP_035478_1775_RED.JP2
ESP_037086_2025_RED.JP2
ESP_037232_1770_RED.JP2
ESP_041030_1735_RED.JP2
ESP_041162_1665_RED.JP2
ESP_041373_1660_RED.JP2
ESP_041900_1805_RED.JP2
ESP_042019_1730_RED.JP2
ESP_042085_1795_RED.JP2
ESP_042098_1775_RED.JP2
ESP_042678_1935_RED.JP2
ESP_043021_1775_RED.JP2
ESP_043153_1685_RED.JP2
ESP_043166_1775_RED.JP2
ESP_043974_2090_RED.JP2
ESP_044326_2045_RED.JP2
ESP_045830_1735_RED.JP2
ESP_045870_1650_RED.JP2
ESP_045975_1685_RED.JP2
ESP_046687_1725_RED.JP2
ESP_046964_1665_RED.JP2
ESP_049812_1735_RED.JP2
ESP_050234_1735_RED.JP2
ESP_050300_1735_RED.JP2

ESP_050379_1725_RED.JP2
ESP_052858_1595_RED.JP2
ESP_054703_2225_RED.JP2
ESP_055614_1605_RED.JP2
ESP_055811_2035_RED.JP2
ESP_056867_1695_RED.JP2
ESP_057025_1640_RED.JP2
ESP_057434_1790_RED.JP2
ESP_057777_1790_RED.JP2
ESP_058133_1770_RED.JP2
ESP_058186_1660_RED.JP2
ESP_058489_1785_RED.JP2
ESP_058542_1625_RED.JP2
ESP_059043_1610_RED.JP2
ESP_059544_1650_RED.JP2
ESP_059623_1790_RED.JP2
ESP_059702_1790_RED.JP2
ESP_061680_1985_RED.JP2
PSP_003647_1745_RED.JP2
PSP_004847_1745_RED.JP2
PSP_004913_1735_RED.JP2
PSP_005058_1720_RED.JP2
PSP_005203_1730_RED.JP2
PSP_005414_1735_RED.JP2
PSP_005625_1730_RED.JP2
PSP_005770_1745_RED.JP2
PSP_007022_2175_RED.JP2
PSP_009620_1660_RED.JP2
PSP_009712_1785_RED.JP2
PSP_009910_1690_RED.JP2

Table S5. List of source images used, available at (PDS Geosciences Nodes, 2020)

References

Amazon Machine Learning. (2021). *Model Fit: Underfitting vs. Overfitting—Amazon Machine Learning*. https://docs.aws.amazon.com/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html

Chen, X., Girshick, R., He, K., & Dollar, P. (2019). TensorMask: A foundation for dense object segmentation. *Proceedings of the IEEE International Conference on Computer Vision*, *2019-Octob*, 2061–2069. https://doi.org/10.1109/ICCV.2019.00215

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 248–255.

Dhillon, A., & Verma, G. K. (2020). Convolutional neural network: A review of models, methodologies and applications to object detection. *Progress in Artificial Intelligence*, *9*(2), 85–112.

Facebook AI Research. (2020). *Welcome to detectron2's documentation! —Detectron2 0.4.1 documentation*. https://detectron2.readthedocs.io/en/latest/index.html

Ghorbanzadeh, O., Blaschke, T., Gholamnia, K., Meena, S., Tiede, D., & Aryal, J. (2019). Evaluation of Different Machine Learning Methods and Deep-Learning Convolutional Neural Networks for Landslide Detection. *Remote Sensing*, *11*(2), 196. https://doi.org/10.3390/rs11020196

Girshick, R. (2015). Fast r-cnn. *Proceedings of the IEEE International Conference on Computer Vision*, 1440–1448.

Google Developers. (2021). *Reducing Loss: Learning Rate | Machine Learning Crash Course*. Google Developers. https://developers.google.com/machine-learning/crash-course/reducing-loss/learning-rate?hl=it

Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., & Chen, T. (2018). Recent advances in convolutional neural networks. *Pattern Recognition*. https://doi.org/10.1016/j.patcog.2017.10.013

He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2018). Mask R-CNN. *ArXiv:1703.06870 [Cs]*. http://arxiv.org/abs/1703.06870

HiRISE | About HiRISE Digital Terrain Models. (2021). https://www.uahirise.org/dtm/about.php

Hoeser, T., & Kuenzer, C. (2020). Object detection and image segmentation with deep learning on Earth observation data: A review-part I: Evolution and recent trends. *Remote Sensing*, *12*(10). https://doi.org/10.3390/rs12101667

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *ArXiv:1704.04861 [Cs]*. http://arxiv.org/abs/1704.04861

Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., & Dollár, P. (2015). Microsoft COCO: Common Objects in Context. *ArXiv:1405.0312 [Cs]*. http://arxiv.org/abs/1405.0312

Massa, F., & Girshick, R. (2018). *maskrcnn-benchmark: Fast, modular reference implementation of Instance Segmentation and Object Detection algorithms in PyTorch*. https://github.com/facebookresearch/maskrcnn-benchmark/

MAY 6, J. S., & Read, 2020 6 Min. (2020, May 6). *What is Mean Average Precision (mAP) in Object Detection?* Roboflow Blog. https://blog.roboflow.com/mean-average-precision/

Merkel, D. (2014). Docker: Lightweight linux containers for consistent development and deployment. *Linux Journal*, *2014*(239), 2.

Neyshabur, B., Sedghi, H., & Zhang, C. (2021). What is being transferred in transfer learning? *ArXiv:2008.11687 [Cs, Stat]*. http://arxiv.org/abs/2008.11687

Neyshabur, B., Zhang, C., & Sedghi, H. (2020). *What is being transferred in transfer learning?*

Nodjoumi, G. (2021, November 29). *DeepLandforms: First Release*. https://doi.org/10.5281/zenodo.5734621

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 32* (pp. 8024–8035). Curran Associates, Inc. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

PDS Geosciences Nodes. (2020). *PDS Geosciences Node Orbital Data Explorer (ODE)*. https://ode.rsl.wustl.edu/

Pham, M.-T., Courtrai, L., Friguet, C., Lefèvre, S., & Baussard, A. (2020). YOLO-Fine: One-Stage Detector of Small Objects Under Various Backgrounds in Remote Sensing Images. *Remote Sensing*, *12*(15), 2501. https://doi.org/10.3390/rs12152501

Ren, S., He, K., Girshick, R., & Sun, J. (2016). Faster R-CNN: towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(6), 1137–1149.

Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., & Liu, C. (2018). A Survey on Deep Transfer Learning. *ArXiv:1808.01974 [Cs, Stat]*. http://arxiv.org/abs/1808.01974

Targ, S., Almeida, D., & Lyman, K. (2016). Resnet in resnet: Generalizing residual architectures. *ArXiv Preprint ArXiv:1603.08029*.

Tempfli, K., Huurneman, G. C., Bakker, W. H., Janssen, L. L. F., Feringa, W. F., Gieske, A. S. M., Grabmaier, K. A., Hecker, C. A., & Horn, J. A. van der. (2009). *Principles of remote sensing: An introductory textbook.* ITC.

Weiss, K., Khoshgoftaar, T. M., & Wang, D. (2016). A survey of transfer learning. *Journal of Big Data*, *3*(1), 9. https://doi.org/10.1186/s40537-016-0043-6

Wu, Y., Kirillov, A., Massa, F., Lo, W.-Y., & Girshick, R. (2019a). *Detectron2 is FAIR's next-generation platform for object detection, segmentation and other visual recognition tasks.* https://github.com/facebookresearch/detectron2

Wu, Y., Kirillov, A., Massa, F., Lo, W.-Y., & Girshick, R. (2019b). *Detectron2/MODEL_ZOO.md at main · facebookresearch/detectron2*. https://github.com/facebookresearch/detectron2/blob/main/MODEL_ZOO.md

Wu, Y., Kirillov, A., Massa, F., Lo, W.-Y., Girshick, R., & Facebook, Inc. (2020). *facebookresearch/detectron2: Detectron2 is FAIR's next-generation platform for object detection and segmentation*. https://github.com/facebookresearch/detectron2

Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2021). 4.4.3 Underfitting or Overfitting? In *Dive into Deep Learning* (pp. 146–147). http://arxiv.org/abs/2106.11342