Segmentation Dataset

In [ ]:
```
import os
import csv

from PIL.Image import Image
import numpy as np

from segwork.data import SegmentationDataset
```

In [ ]:
```
class DroneDataset(SegmentationDataset):
    """Dataset for Semantic Drone dataset

    The Semantic Drone Dataset focuses on semantic understanding of urban scenes for
    increasing the safety of autonomous drone flight and landing procedures.
    The imagery depicts  more than 20 houses from nadir (bird's eye) view acquired at an
    altitude of 5 to 30 meters above ground. A high resolution camera was used to acquire
    images at a size of 6000x4000px (24Mpx). The training set contains 400 publicly available
    images and the test set is made up of 200 private images.

    https://www.tugraz.at/index.php?id=22387"""

    HEIGHT = 4000
    WIDTH = 6000

    def __init__(self, pil_target:bool = True, *args, **kwargs):
        super().__init__(*args, **kwargs)
        _TRAINING_DIR = os.path.join(self.root,'training_set')
        self.TRAINING_IMAGES_DIR = os.path.join(_TRAINING_DIR, 'images')
        self.TRAINING_SEMANTICS = os.path.join(_TRAINING_DIR, 'gt', 'semantic')
        self.TRAINING_LABELS_DIR = os.path.join(self.TRAINING_SEMANTICS, 'label_images')
        self.TRAINING_LABELS_DIR_NUMPY = os.path.join(self.TRAINING_SEMANTICS, 'label_numpy')
        self.pil_target = pil_target

    @property
    def images(self):
        data_dir = self.TRAINING_LABELS_DIR if self.split == 'train' else self.TRAINING_LABELS_DIR
        return self._get_listdir(data_dir)
```

```
    def load_image(self, idx:int):
        return Image.open(self.images[idx]).convert("RGB")

    @property
    def annotations(self):
        data_dir = self.TRAINING_LABELS_DIR
        return self._get_listdir(data_dir)

    def _get_listdir(self, dir:str):
        """Return a list with the path to the files in it"""
        return [os.path.join(dir, file) for file in os.listdir(dir)]

    def load_label(self, idx:int):
        if self.pil_target:
            return Image.open(self.annotations[idx]).convert("RGB")
        return self.load_numpy_label(idx)

    @property
    def mask_colors(self):
        with open(os.path.join(self.TRAINING_SEMANTICS, 'class_dict.csv' ), 'r') as csvfile:
            reader = csv.reader(csvfile)
            return { tuple([int(r.strip()),int(g.strip()),int(b.strip())]) : name for (name, r, g, b) in reader  }

    @property
    def mask_colors_index(self):
        return { key : idx for idx, key in enumerate(self.mask_colors)}

    @property
    def classes(self):
        return list(self.mask_colors.values())

    def load_numpy_label(self, idx:int, *args, **kwargs):
        """Return a :py:class:`numpy.ndarray` with the label for the specified idx"""
        file_name = f'{idx:03d}.npy'
        path_name = os.path.join(self.TRAINING_LABELS_DIR_NUMPY, file_name)
        return np.load(path_name, *args, **kwargs)

    def load_weight_label(self, idx):
        """Load label to be used by the calculator"""
        return self.load_numpy_label(idx)
```

In [ ]:

```
DATA_DIR = os.path.join('data')
dataset = DroneDataset(
```

```
    root = os.path.join(DATA_DIR, 'semantic_drone_dataset'),
    pil_target=True,
)
```

In [ ]:
```
dataset
```

Out[ ]:
```
Dataset DroneDataset
    Number of datapoints: 400
    Root location: data\semantic_drone_dataset
```