MMsegmentation with a custom dTag-dataset

This tutorial will guide you through the process of training a deep neural network for semantic segmentation using the open-source framework MMSegmentation and a custom dTag dataset in **COCO format**.

This tutorial shows only one of many ways in which you can train an MMSegmentation model with your own dataset. Which option you choose depends on the structure of your dataset.

Requirements for following this tutorial:

Basic knowledge and understanding of Deep Learning, Python and virtual environments

AND

a computer with a GPU!

---

0. Installation

1

Download and install MMSegmentation

To download and install MMSegmentation locally you either need to follow the official instructions **or** the steps explained below:

1. Create a new **conda** environment (install from official website) by running the following commands in your terminal:

```
conda create --name openmmlab python=3.8 -y
conda activate openmmlab
```

2. Install PyTorch following the official instructions.
3. Run the following commands:

```
pip install -U openmim
mim install mmengine
mim install "mmcv>=2.0.0"
```

4. Install MMSegmentation:

```
git clone -b main https://github.com/open-mmlab/mmsegmentation.git
cd mmsegmentation
pip install -v -e .
pip install -r requirements.txt
```

After the successful installation, you should now have a new folder named "mmsegmentation" in your file system. The folder should look something like this.

2

Prepare dataset folder

Under the newly installed folder "mmsegmentation" create a new folder called "data". This is the folder in which your dataset(s) will later be stored.

```
mkdir data
```

1. Dataset preparation

1

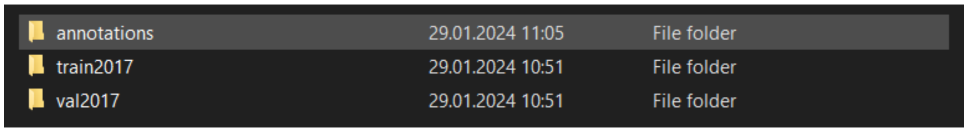
Export from dTag

1. Go to the dTag domain you want to export.
2. In the sidebar menu click on "Export".
3. In the top right corner click on "Create" and choose "Object detection" from the dropdown menu.
4. Configure the export the way you want it. **Important:**
   a. Under "Features" make sure to add every segmentation layer individually (should look like this)
   b. Under "Export Settings Splits" add a **maximum of two** splits. Adding more than two splits would require additional configuration steps.
   The splits should be called "train2017" and "val2017".
   In the end the "Splits" section should look like this.
5. Click on "Run export".
6. After the export has finished, click on "Get Downloads" and download the file "coco.zip".

2

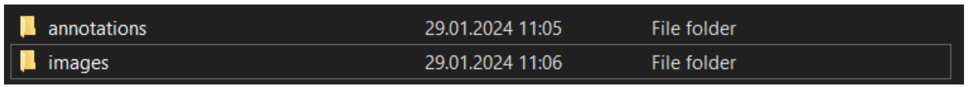
Edit file structure

1. Under "mmsegmentation/data" create a new folder with the name of your dataset (e.g. "waste")
2. Go to your downloads folder and unzip the "coco.zip" file to "mmsegmentation/data/<your-dataset-name>".
   This should create three folders "annotations", "train2017" and "val2017".
   The folder "mmsegmentation/data/<your-dataset-name> should now look like this:



annotations	29.01.2024 11:05	File folder
train2017	29.01.2024 10:51	File folder
val2017	29.01.2024 10:51	File folder

3. Create a new folder "images" and move the folders "train2017" and "val2017" into this folder.
   The new file structure should look like this:



annotations	29.01.2024 11:05	File folder
images	29.01.2024 11:06	File folder

3

Convert dataset format

1. Download this script.
2. Open your terminal and activate the conda environment you created in step 0.1

```
conda activate openmmlab
```

3. Install the following packages:

```
pip install pycocotools
pip install pillow
pip install numpy
pip install argparse
```

4. Run the script with the command:

   ```
   python /path/to/the/script.py /path/to/mmsegmentation/data/your-dataset-name/annotations
   /folder/
   ```

   This will convert all images in your dataset folder according to the annotation files in the annotations folder.
5. After running the script successfully, there should be two new folders inside of the annotations folder: "train2017" and "val2017".
   The images in these folders might appear black!

6. Go to the folder "mmsegmentation/data/your-dataset-name/annotations" and delete all .json-files.

7. The final folder structure should look like this:

   ```
   mmsegmentation
   | - ...
   | - data
   |     | - <your-dataset-name>
   |     |     | - annotations
   |     |     |     | - train2017
   |     |     |     | - val2017
   |     |     | - images
   |     |     |     | - train2017
   |     |     |     | - val2017
   | - ...
   ```

2. Create configuration files

To be able to use your own custom dataset in MMSegmentation, you need to create/edit four different configuration files in total.

1

Create new dataset class

Under "mmsegmentation/mmseg/datasets/" create a new python file "<your-dataset-name>.py".

In this file you will need to specify the new dataset class as shown in the example below.

```
from mmseg.registry import DATASETS
from .basesegdataset import BaseSegDataset

@DATASETS.register_module()
class WasteDataset(BaseSegDataset):

    METAINFO = dict(
                # All segmentation classes of the dataset
        classes = (
            'Aluminium',
            'Belt',
            'Corrugated cardboard',
            'Gray cardboard',
            'Newspaper',
            'Other',
            'Paper',
            'Plastic',
            'Tetra'
        ),
                # One colour (RGB) per segmentation class. Used to visualize the prediction result.
        palette=[
            [0, 0, 128],
            [9, 4, 15],
            [254, 224, 27],
            [0, 130, 200],
            [128, 0, 0],
            [245, 130, 48],
            [165, 0, 189],
            [57, 186, 173],
            [0, 89, 0]
        ]
    )

    def __init__(self,
                 img_suffix='.png',
                 seg_map_suffix='_labelTrainIds.png',
                 **kwargs) -> None:
        super().__init__(img_suffix=img_suffix, seg_map_suffix=seg_map_suffix, **kwargs)
```

To create your own class you will need to change the variables "classes", "palette", "img_suffix" and "seg_map_suffix".

2

Register dataset class

After creating the dataset class, the next step is to register this class under "mmsegmentation/mmseg/datasets/__init__.py".

To register your class you´ll need to add the following line to the existing file:

```
from .<filename-of-your-class> import <your-class-name>
```

Furthermore, you´ll have to append the class name to the "__all__" variable in the same file.

3

Create dataset configuration file

Next up, you´ll need to create a new dataset configuration file under "mmsegmentation/configs/_base_/datasets/<your-config-file>.py". This file specifies how MMSegmentation is supposed to handle the new dataset (for example: where to find it).

To create your configuration file you can copy the example code below and change the necessary variables:

```
# dataset settings
dataset_type = 'WasteDataset' #Substitute with your dataset class name
data_root = 'data/waste'      #Substitute with the path to your dataset
crop_size = (241, 323)
train_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='LoadAnnotations'),
    dict(
        type='RandomResize',
        scale=(2048, 512),
        ratio_range=(0.5, 2.0),
        keep_ratio=True),
    dict(type='RandomCrop', crop_size=crop_size, cat_max_ratio=0.75),
    dict(type='RandomFlip', prob=0.5),
    dict(type='PhotoMetricDistortion'),
    dict(type='PackSegInputs')
]
test_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='Resize', scale=(2048, 512), keep_ratio=True),
    # add loading annotation after ``Resize`` because ground truth
    # does not need to do resize data transform
    dict(type='LoadAnnotations'),
    dict(type='PackSegInputs')
]
img_ratios = [0.5, 0.75, 1.0, 1.25, 1.5, 1.75]
tta_pipeline = [
    dict(type='LoadImageFromFile', backend_args=None),
    dict(
        type='TestTimeAug',
        transforms=[
            [
                dict(type='Resize', scale_factor=r, keep_ratio=True)
                for r in img_ratios
            ],
            [
                dict(type='RandomFlip', prob=0., direction='horizontal'),
                dict(type='RandomFlip', prob=1., direction='horizontal')
            ], [dict(type='LoadAnnotations')], [dict(type='PackSegInputs')]
        ])
]
train_dataloader = dict(
    batch_size=4,
    num_workers=4,
    persistent_workers=True,
    sampler=dict(type='InfiniteSampler', shuffle=True),
    dataset=dict(
        type=dataset_type,
        data_root=data_root,
        data_prefix=dict(
            img_path='images/train2017', seg_map_path='annotations/train2017'),
        pipeline=train_pipeline))
val_dataloader = dict(
    batch_size=1,
    num_workers=4,
    persistent_workers=True,
    sampler=dict(type='DefaultSampler', shuffle=False),
    dataset=dict(
        type=dataset_type,
        data_root=data_root,
        data_prefix=dict(
            img_path='images/val2017', seg_map_path='annotations/val2017'),
        pipeline=test_pipeline))
test_dataloader = val_dataloader

val_evaluator = dict(type='IoUMetric', iou_metrics=['mIoU'])
test_evaluator = val_evaluator
```

4

Create model configuration file

The last file you´ll need to create is a model configuration file. Before you create the file you´ll have to choose the model you want to use for training. You can find the available models under "mmsegmentation/configs/_base_/models/".

After choosing a model, for example "DeepLabV3", the new configuration file needs to be created in the folder "mmsegmentation /configs/<model-name>/<config-file>.py".

When creating the config file it is highly recommended to keep to the official naming convention.

The last step is to copy the example code below, change the "__base__" variable according to your needs and paste it into the new config file.

```
_base_ = [
    '../_base_/models/deeplabv3_r50-d8.py',    # Path to the model
    '../_base_/datasets/waste_241x323.py',     # Path to your dataset config file
    '../_base_/default_runtime.py',
    '../_base_/schedules/schedule_20k.py'      # Specify according to your needs
]
crop_size = (241, 323)
data_preprocessor = dict(size=crop_size)
model = dict(
    data_preprocessor=data_preprocessor,
    decode_head=dict(num_classes=9),
    auxiliary_head=dict(num_classes=9))
```

After creating this file you can now start training the model!

3. Training and inference

1

Training

1. Open the Terminal, activate the conda environment and navigate to the "mmsegmentation" folder.
2. Run the following command:

```
python tools/train.py configs/your_model/your_model_config.py
```

3. After training was successful, a new folder "mmsegmentation/work_dirs/" and a subfolder "work_dirs /your_model_config_name/" should have been automatically created.
   Among other things the subfolder contains the model config file and the checkpoint files created during training (for example: "iter_20000.pth").

2

Inference

To use the newly trained model for inference create a python file in the "work_dirs" subfolder called "<your-dataset-name>_inference.py" and paste the example code below into it (you´ll need to change the corresponding variables).

```
import mmcv
from mmseg.apis import init_model, inference_model, show_result_pyplot

checkpoint_path = './work_dirs/deeplabv3_r50-dg_4xb4-20k_waste-241x323/iter_2000.
pth'                                # Path to a checkpoint file for the prediction
config_file = './work_dirs/deeplabv3_r50-dg_4xb4-20k_waste-241x323/deeplabv3_r50-dg_4xb4-20k_waste-
241x323.py'       # Path to your model config file

img = mmcv.imread('./data/waste/images/train2017/00078c01-2861-4baf-ae9f-72da83893368.
png')                           # Path to an image to perform the inference on
palette = [[0, 0, 128],[9, 4, 15],[254, 224, 27],[0, 130, 200],[128, 0, 0],[245, 130, 48],[165, 0,
189],[57, 186, 173],[0, 89, 0]]    # Colour palette to vizualize the results (can be copied from
your dataset class)

model = init_model(config_file, checkpoint_path, 'cuda:0')
result = inference_model(model, img)
vis_result = show_result_pyplot(model, img, result, show=False, save_dir='./work_dirs/results
/')                     # Path to the result output
```

Then open your terminal, activate the conda environment and navigate to the "mmsegmentation" folder.

After doing these steps, performing inference becomes as easy as running the following command in the terminal:

```
python /path/to/inference/script.py
```

If all goes well there should be a new image in the "results" path (specified in "<your-dataset-name>_inference.py") with the inference prediction in colour. The image might look something like below: