Using OpenMMLab to Accelerate Your Research

Wenwei Zhang

Outline

1. Run on customized dataset: nuImages Dataset
2. MIM: **M**IM **I**nstalls Open**MM**Lab projects
3. Develop a new algorithm: Swin Transformer

MIM Examples

Run on Customized Dataset

Key steps:

1. Preprocess the dataset
2. Implement a new dataset class
3. Modify config file to use it
4. Train and test a model

Run on Customized Dataset

Key steps:

1. Preprocess the dataset
2. Implement a new dataset class
3. Modify config file to use it
4. Train and test a model

File structure:

Tiny and decoupled from MMSegmentation

```
project
├── configs
│   ├── _base_
│   │   ├── datasets
│   │   │   └── nuimages.py
│   │   ├── default_runtime.py
│   │   ├── models
│   │   │   └── pspnet_r50-d8.py
│   │   └── schedules
│   │       └── schedule_80k.py
│   └── pspnet
│       └── pspnet_r18-d8_512x1024_80k_nuim.py
├── nuim_converter.py
└── nuim_dataset.py
```

Run on Customized Dataset

Key steps:

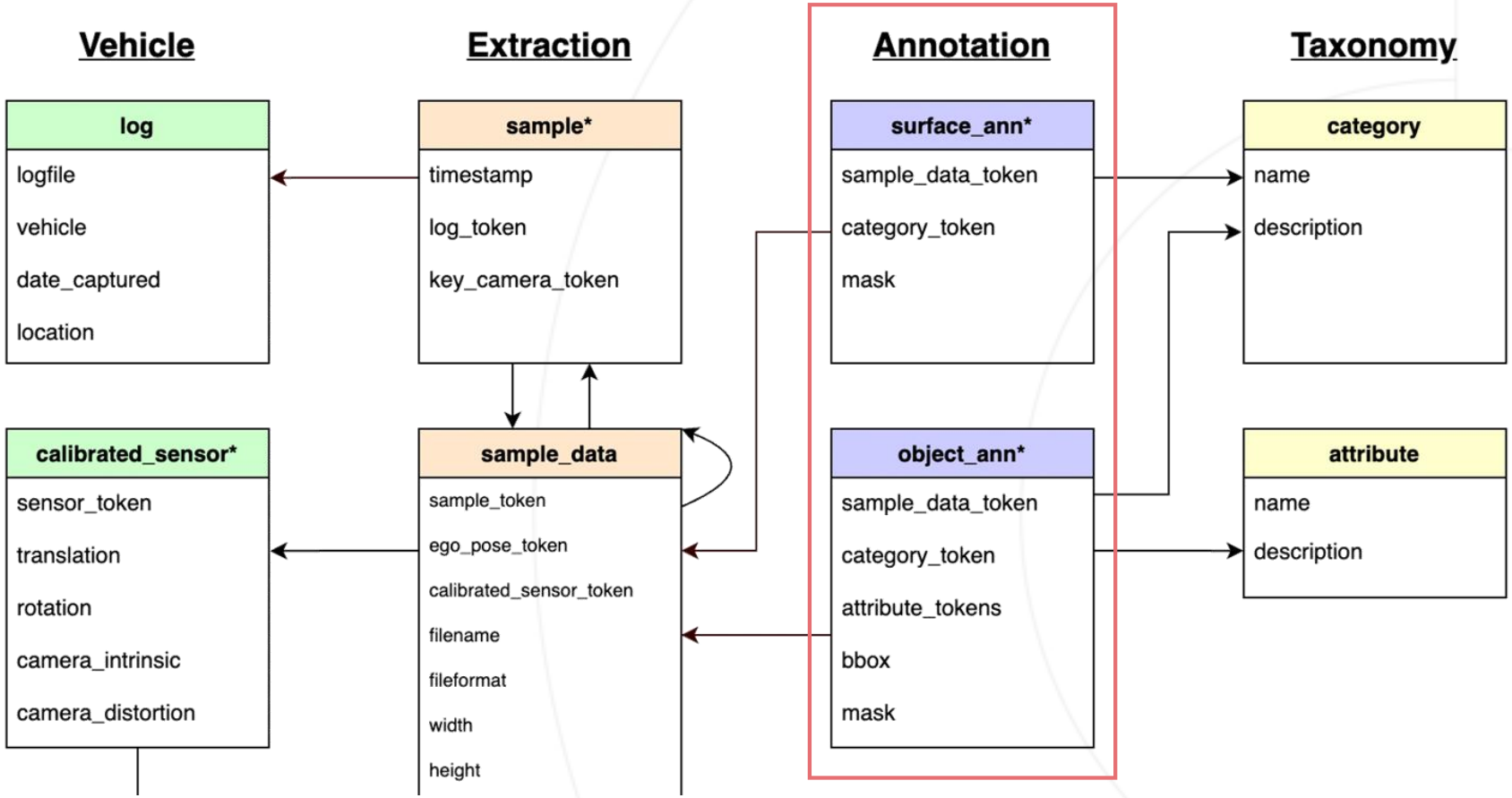
1. Preprocess the dataset
2. Implement a new dataset class
3. Modify config file to use it
4. Train and test a model

File structure:

```
project
├── configs
│   ├── _base_
│   │   ├── datasets
│   │   │   └── nuimages.py
│   │   ├── default_runtime.py
│   │   ├── models
│   │   │   └── pspnet_r50-d8.py
│   │   └── schedules
│   │       └── schedule_80k.py
│   └── pspnet
│       └── pspnet_r18-d8_512x1024_80k_nuim.py
├── nuim_converter.py
└── nuim_dataset.py
```

Run on Customized Dataset

1. Preprocess the dataset



Screenshot from https://www.nuscenes.org/nuimages

Run on Customized Dataset

1. Preprocess the dataset

```
python -u nuim_converter.py \
    --data-root $DATA \
    --versions $VERSIONS \
    --out-dir $OUT \
    --nproc $NUM_WORKERS
```

```
project
├── nuimages
│   ├── annotations
│   │   ├── nuimages_v1.0-mini.json
│   │   ├── nuimages_v1.0-train.json
│   │   ├── nuimages_v1.0-val.json
│   │   ├── nuimages_v1.0-val2400.json
│   │   ├── nuimages_v1.0-test.json
│   │   └── semantic_masks
│   │       ├── xxxxx.png
│   │       └── xxxxx.png
│   └── samples
```

Run on Customized Dataset

Key steps:

1. Preprocess the dataset
2. Implement a new dataset class
3. Modify config file to use it
4. Train and test a model

File structure:

```
project
├── configs
│   ├── _base_
│   │   ├── datasets
│   │   │   └── nuimages.py
│   │   ├── default_runtime.py
│   │   ├── models
│   │   │   └── pspnet_r50-d8.py
│   │   └── schedules
│   │       └── schedule_80k.py
│   └── pspnet
│       └── pspnet_r18-d8_512x1024_80k_nuim.py
├── nuim_converter.py
└── nuim_dataset.py
```

Run on Customized Dataset

2. Implement a new dataset class

```python
import os.path as osp

import mmcv
from mmcv.utils import print_log
from mmseg.datasets import CustomDataset
from mmseg.datasets.builder import DATASETS
from mmseg.utils import get_root_logger


@DATASETS.register_module()
class NuImagesDataset(CustomDataset):
    CLASSES = ()

    def load_annotations(self, img_dir, img_suffix, ann_dir,
                         seg_map_suffix, split):

        annotations = mmcv.load(split)
        img_infos = []
        for img in annotations['images']:
            img_info = dict(filename=img['file_name'])
            seg_map = img_info['filename'].replace(
                img_suffix, seg_map_suffix)
            img_info['ann'] = dict(
                seg_map=osp.join('semantic_masks', seg_map))
            img_infos.append(img_info)

        print_log(
            f'Loaded {len(img_infos)} images from {ann_dir}',
            logger=get_root_logger())
        return img_infos
```

Use MMSegmentation as a library like PyTorch and torchvision

Register the dataset into the DATASETS registry

Inherits from CustomDataset

Only need to override the load_annotations function to convert the annotations into middle format in MMSegmentation

Run on Customized Dataset

Key steps:

1. Preprocess the dataset
2. Implement a new dataset class
3. Modify config file to use it
4. Train and test a model

File structure:

```
project
├── configs
│   ├── _base_
│   │   ├── datasets
│   │   │   └── nuimages.py
│   │   ├── default_runtime.py
│   │   ├── models
│   │   │   └── pspnet_r50-d8.py
│   │   └── schedules
│   │       └── schedule_80k.py
│   └── pspnet
│       └── pspnet_r18-d8_512x1024_80k_nuim.py
├── nuim_converter.py
└── nuim_dataset.py
```

Run on Customized Dataset

3. Modify config file to use it

```
project
├── configs
│   ├── _base_
│   │   ├── datasets
│   │   │   └── nuimages.py
│   │   ├── default_runtime.py
│   │   ├── models
│   │   │   └── pspnet_r50-d8.py
│   │   └── schedules
│   │       └── schedule_80k.py
│   └── pspnet
│       └── pspnet_r18-d8_512x1024_80k_nuim.py
├── nuim_converter.py
└── nuim_dataset.py
```

A complete config file is decomposed into four basic configs for free combination without writing duplicated code.

Run on Customized Dataset

3. Modify config file to use it

Base config of nuImages dataset

```
dataset_type = 'NuImagesDataset'
data_root = 'data/nuimages/'
train_pipeline = [
    ...
]
test_pipeline = [
    ...
]
data = dict(
    samples_per_gpu=2,
    workers_per_gpu=2,
    train=dict(
        type=dataset_type,
        data_root=data_root,
        img_dir='',
        ann_dir='annotations/',
        split='annotations/nuimages_v1.0-train.json',
        pipeline=train_pipeline),
    val=dict(
        type=dataset_type,
        ...),
    test=dict(
        type=dataset_type,
        ...))

custom_imports = dict(
    imports=['nuim_dataset'],
    allow_failed_imports=False)
```

Define data pipeline of the dataset

Make the file imported so that nuImagesDataset can be registered

Config to run PSPNet-R18 on nuImages dataset

```
_base_ = [
    '../_base_/models/pspnet_r50-d8.py',
    '../_base_/datasets/nuimages.py',
    '../_base_/default_runtime.py',
    '../_base_/schedules/schedule_80k.py'
]

model = dict(
    pretrained='open-mmlab://resnet18_v1c',
    backbone=dict(depth=18),
    decode_head=dict(
        in_channels=512,
        channels=128,
        num_classes=25),
    auxiliary_head=dict(
        in_channels=256,
        channels=64,
        num_classes=25))

# use cityscapes pre-trained models
load_from = (
    'https://download.openmmlab.com/mmsegmentation/v0.5/pspnet/'
    'pspnet_r18-d8_512x1024_80k_cityscapes/'
    'pspnet_r18-d8_512x1024_80k_cityscapes_20201225_021458-09ffa746.pth')
```

Use base configs to for simplicity

Only override the necessary keys

Use Cityscapes Pretrained model

Run on Customized Dataset

4. Train and test the model

Train the model

```
PYTHONPATH='.'$PYTHONPATH mim train mmseg \
    configs/pspnet/pspnet_r18-d8_512x1024_80k_nuim.py
    --work-dir $WORK_DIR \
    --launcher slurm -G 8 -p $PARTITION
```

Test the trained model

```
PYTHONPATH='.'$PYTHONPATH mim test mmseg \
    configs/pspnet/pspnet_r18-d8_512x1024_80k_nuim.py
    --checkpoint $WORK_DIR/latest.pth \
    --launcher slurm -G 8 -p $PARTITION \
    --eval mIoU
```

Wait…. What is MIM?

MIM： MIM Installs OpenMMLab Packages

Package Management

- Infer mmcv-full pre-build package automatically
- Handle dependencies of OpenMMLab projects

```
pip install openmim>=0.1.1  # install mim through pypi
mim install mmcv-full==1.3.5
mim install mmdet==2.13.0
mim install mmsegmentation=0.14.0
```

Unified Entrypoint for Scripts

- mim train mmdet ⟶ mmdetection/tools/train.py
- mim train mmseg ⟶ mmsegmentation/tools/train.py
- mim train mmcls ⟶ mmclassification/tools/train.py

MIM: MIM Installs OpenMMLab Packages

Model Management

Use a similar format to store model information as Paper with Code (PWC)

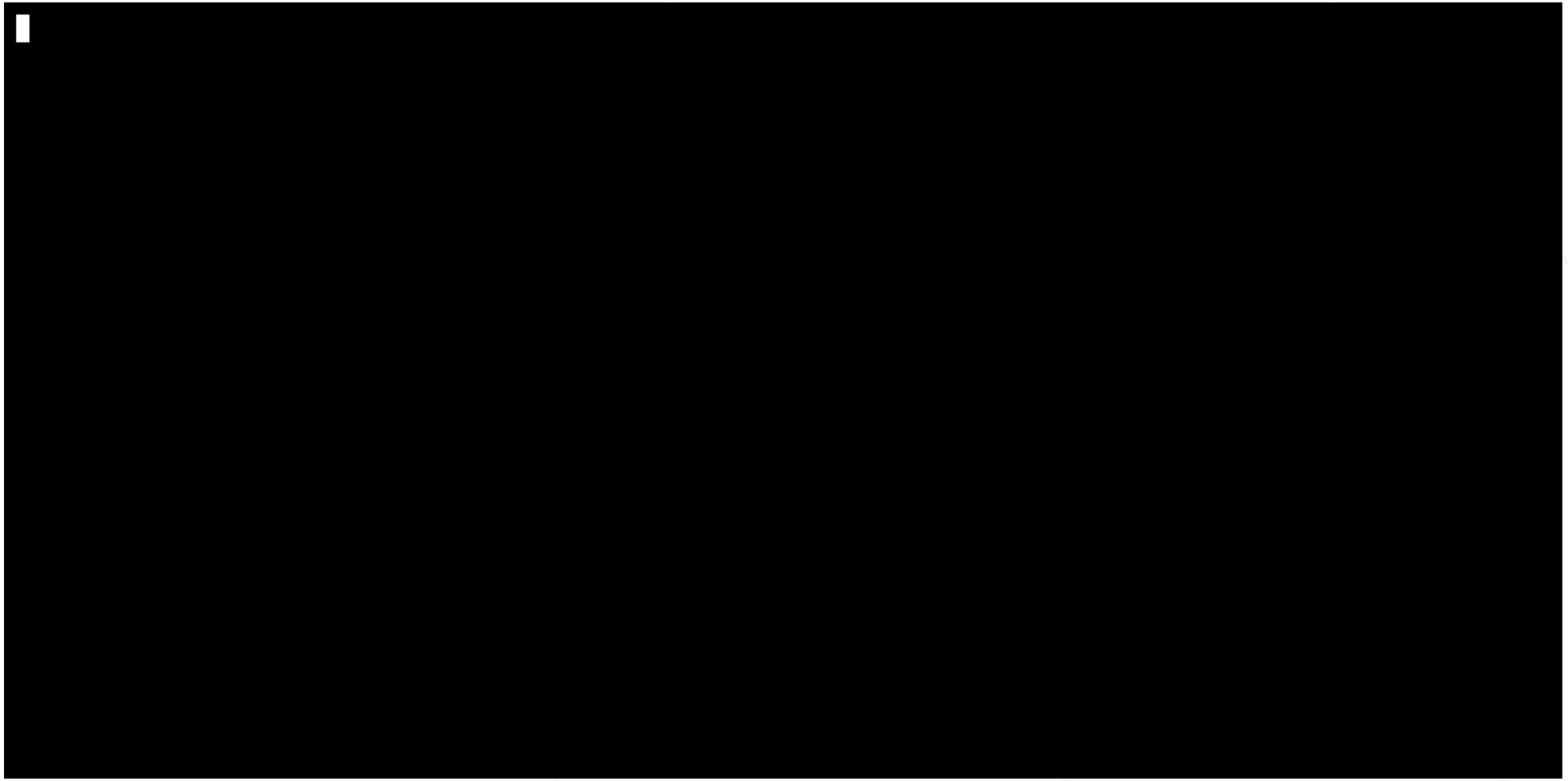
You can

1. download models by name
2. search models that meet specific criteria

Find out more at

MIM： MIM Installs OpenMMLab Packages

Examples of MIM



Find out more at

Develop a New Algorithm

Key steps:

1. Implement the Swin Transformer and register Swin Transformer into registry
2. Modify config file to use it
3. Train and test a model

File structure:

A minimal implementation of Swin Transformer

```
├── configs
│   ├── swin_classifier
│   │   └──swin_tiny_224_imagenet.py
│   ├── swin_mask_rcnn
│   │   └──mask_rcnn_swim-t-p4-w7_fpn_1x_coco.py
│   └── swin_upernet
│       └── upernet_swin-t_512x512_160k_8x2_ade20k.py
└── swin
    └── swin_transformer.py
```

Develop a New Algorithm

Key steps:

1. Implement the Swin Transformer and register Swin Transformer into registry
2. Modify config file to use it
3. Train and test a model

File structure:

A minimal implementation of Swin Transformer

```
├── configs
│   ├── swin_classifier
│   │   └──swin_tiny_224_imagenet.py
│   ├── swin_mask_rcnn
│   │   └──mask_rcnn_swim-t-p4-w7_fpn_1x_coco.py
│   └── swin_upernet
│       └── upernet_swin-t_512x512_160k_8x2_ade20k.py
└── swin
    └── swin_transformer.py
```

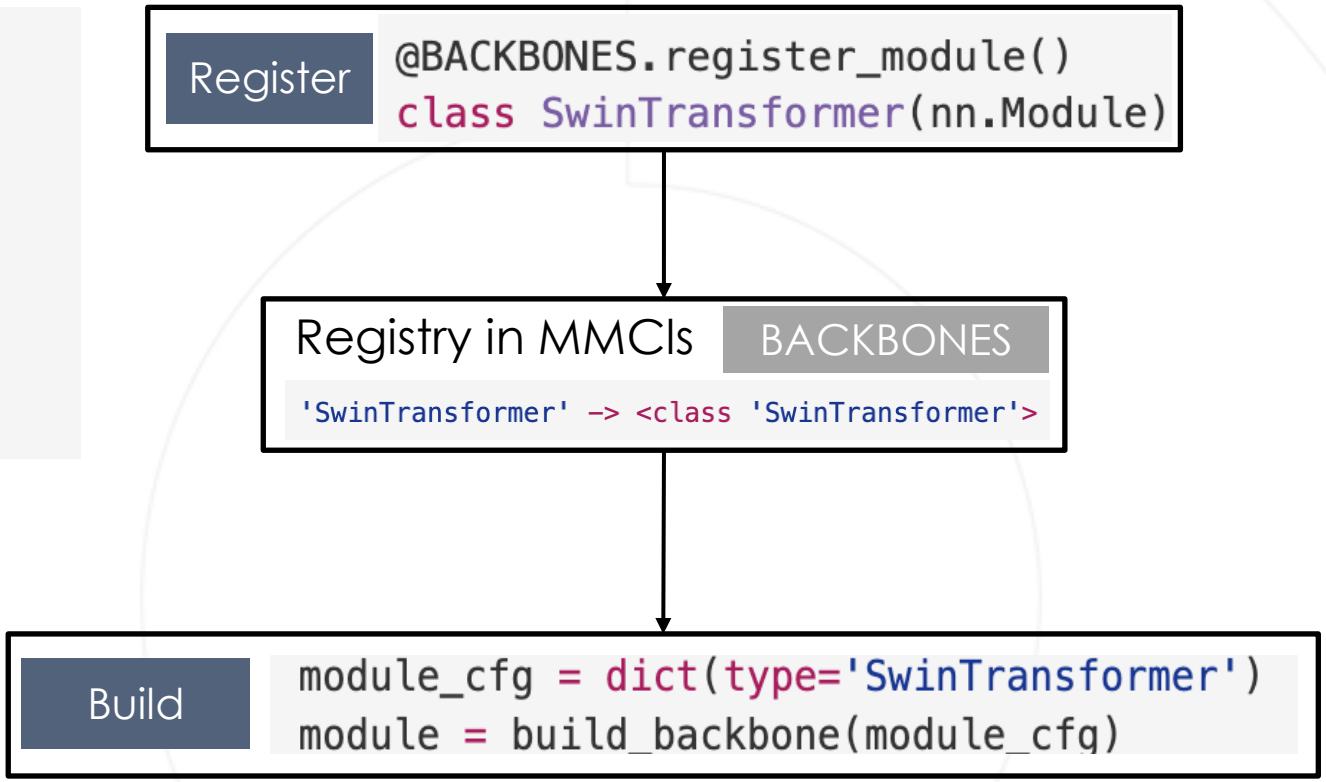
Develop a New Algorithm

OpenMMLab

Step 1: implement Swin transformer

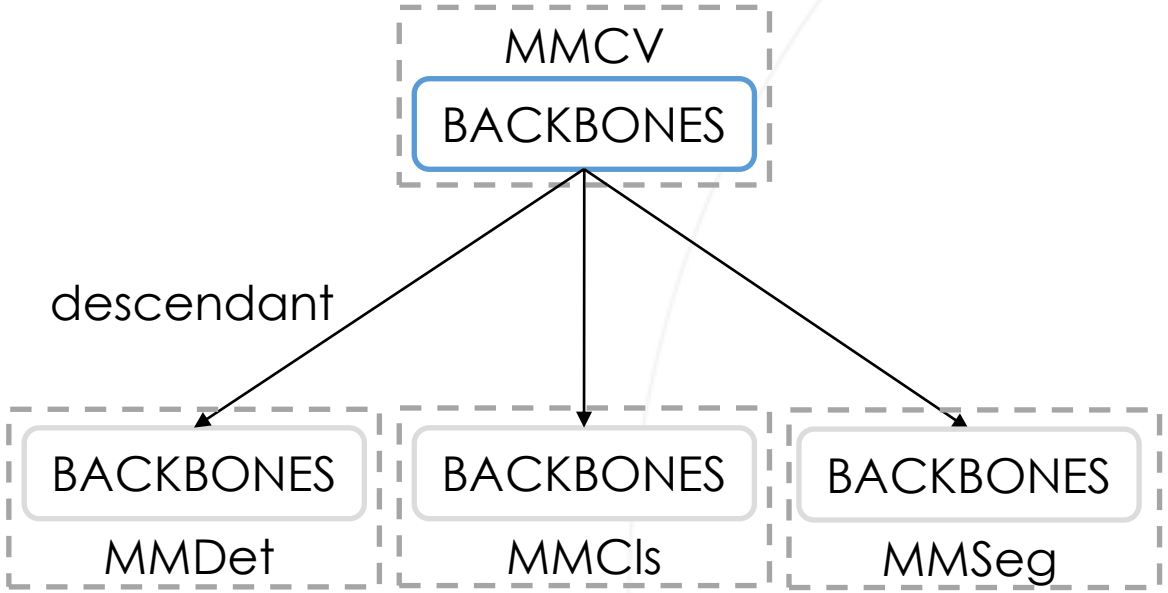
```python
from mmcls.models import BACKBONES


@BACKBONES.register_module()
class SwinTransformer(nn.Module):
    # code implementation
    def __init__(self, *args, **kwargs):
        super().__init__()
```



Develop a New Algorithm

Structure of registry



Register

```
@BACKBONES.register_module()
class SwinTransformer(nn.Module)
```

Registry in MMCls BACKBONES

```
'SwinTransformer' -> <class 'SwinTransformer'>
```

Registry in MMDet BACKBONES

```
'mmcls.SwinTransformer' -> <class 'SwinTransformer'>
```

Build

```
module_cfg = dict(type='mmcls.SwinTransformer')
module = build_backbone(module_cfg)
```

Develop a New Algorithm

Key steps:

1. Implement the Swin Transformer and register Swin Transforemer into registry
2. Modify config files to use it
3. Train and test a model

File structure:

A minimal implementation of Swin Transformer

```
├── configs
│   ├── swin_classifier
│   │   └──swin_tiny_224_imagenet.py
│   ├── swin_mask_rcnn
│   │   └──mask_rcnn_swim-t-p4-w7_fpn_1x_coco.py
│   └── swin_upernet
│       └── upernet_swin-t_512x512_160k_8x2_ade20k.py
└── swin
    └── swin_transformer.py
```

Step 2: Modify config files to use Swin Transformer

Key components in the config for instance segmentation

```
_base_ = [
    '../_base_/models/mask_rcnn_r50_fpn.py',
    '../_base_/datasets/coco_instance.py',
    '../_base_/schedules/schedule_1x.py',
    '../_base_/default_runtime_det.py'
]

model = dict(
    pretrained='./pretrain/swin/swin_tiny_patch4_window7_224.pth',
    backbone=dict(type='mmcls.SwinTransformer'))

custom_imports = dict(
    imports=['swin.swin_transformer'], allow_failed_imports=False)
```

Make the file imported so that SwinTransformer can be registered

Step 2: Modify config files to use Swin Transformer

Key components in the config for semantic segmentation

```
_base_ = [
    '../_base_/models/upernet_r50.py',
    '../_base_/datasets/ade20k.py',
    '../_base_/default_runtime_seg.py',
    '../_base_/schedules/schedule_160k.py'
]

model = dict(
    pretrained='./pretrain/swin/swin_tiny_patch4_window7_224.pth',
    backbone=dict(type='mmcls.SwinTransformer'))

custom_imports = dict(
    imports=['swin.swin_transformer'], allow_failed_imports=False)
```

Step 2: Modify config files to use Swin Transformer

Key components in the config for image classification

```
_base_ = [
    '../_base_/datasets/imagenet_bs128_swin_224.py',
    '../_base_/schedules/imagenet_bs1024_adamw_swin.py',
    '../_base_/default_runtime.py'
]

model = dict(
    type='ImageClassifier',
    backbone=dict(
        type='SwinTransformer', arch='tiny', img_size=224, drop_path_rate=0.2),
    neck=dict(type='GlobalAveragePooling', dim=1),
    head=dict(
        type='SwinLinearClsHead',
        num_classes=1000,
        in_channels=768,
        loss=dict(type='CrossEntropyLoss', use_soft=True),
        cal_acc=False),
    train_cfg=dict(
        cutmixup=dict(
            mixup_alpha=0.8,
            cutmix_alpha=1.0,
            prob=1.0,
            switch_prob=0.5,
            mode='batch',
            label_smoothing=0.1)))

custom_imports = dict(
    imports=['swin.swin_transformer'], allow_failed_imports=False)
```

OpenMMLab

Step 3: Train the model

Train Swin Mask-RCNN with MMDetection

```
PYTHONPATH='.':$PYTHONPATH mim train mmdet \
    configs/swin_mask_rcnn/mask_rcnn_swim-t-p4-w7_fpn_fp16_1x_coco.py \
    --work-dir ../work_dir/mask_rcnn_swim-t-p4-w7_fpn_fp16_1x_coco.py \
    --launcher slurm --partition $PARTITION -G 8 --gpus-per-node 8 \
    --srun-args $SRUN_ARGS
```

Train Swin UperNet with MMSegmentation

```
PYTHONPATH='.':$PYTHONPATH mim train mmseg \
    configs/upernet/upernet_swin-t_512x512_160k_8x2_ade20k.py \
    --work-dir ../work_dir/upernet_swin-t_512x512_160k_8x2_ade20k.py \
    --launcher slurm --partition $PARTITION -G 8 --gpus-per-node 8 \
    --srun-args $SRUN_ARGS
```

Find out more at

Develop a New Algorithm

Takeaway:

1. Use MIM to launch training and testing in your projects

2. Use registry to freely implement modules and extend OpenMMLab libraries, rather than forking and modifying the library

3. General modules like backbones only need one implementation for multiple tasks by specifying the scopes of the modules

Thank You!