Registry

Model registry - Create registry

In [ ]:
```python
from segwork import ConfigurableRegistry

dataset_reg = ConfigurableRegistry(
    class_key='dataset',
    unique = True,
    additional_args=['transform', 'target_transform'],
)
```

In [ ]:
```python
from segwork.data.drone_dataset import DroneDataset
```

In [ ]:
```python
dataset_reg['drone'] = {
    'dataset': DroneDataset}
```

In [ ]:
```python
dataset_reg
```

Out[ ]:
```
ConfigurableRegistry
        attr_name: _register_name
        unique: True
        Number of registered classes: 1
        Registered classes: ['drone']
        Class key: dataset
        Attribute args: _default_args
        Attribute kwargs: _default_kwargs
        Additional info from attributes: ['transform', 'target_transform']
```

Model registry - Add items to a registry

In [ ]:
```python
import torch.nn as nn
from segwork.model import models_reg
```

In [ ]:
```python
@models_reg.register
class NeuralNetworkDecorated(nn.Module):
    _register_name='Net'

    _default_kwargs = {
        'size' : 28
    }

    def __init__(self, size: int = 28):
        super(NeuralNetworkDecorated, self).__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(size*size, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 10),
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits

class NeuralNetworkDecoratedB(nn.Module):

    _register_name='NetBig'

    _default_kwargs = {
        'size' : 112
    }
```

In [ ]:
```python
models_reg['NetBig'] = {
    'wrong_key': NeuralNetworkDecoratedB
}
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_4840\2406053431.py in <module>
      1 models_reg['NetBig'] = {
----> 2     'wrong_key': NeuralNetworkDecoratedB
      3 }

c:\Users\alvar\Projects\segwork\segwork\registry.py in __setitem__(self, key, value)
    148             key (typing.Hashable): Lookup key.
    149         """
--> 150         self._validate_register(key, value)
    151         self._register(key, value)
    152 

c:\Users\alvar\Projects\segwork\segwork\registry.py in _validate_register(self, key, value)
    168         """Validate register"""
    169         self._validate_key(key)
--> 170         self._validate_value(value)
    171 
    172     def _validate_key(self, key):

c:\Users\alvar\Projects\segwork\segwork\registry.py in _validate_value(self, value)
    376 
    377     def _validate_value(self, value: typing.Dict):
--> 378         assert self._class_key in value, f'Value must have a key {self._class_key} containing a reference to the class.'
    379         # Warning if no args are store.
    380 

AssertionError: Value must have a key model containing a reference to the class.
```

```
In [ ]: models_reg['NetBig'] = {
            'model': NeuralNetworkDecoratedB
        }
```

```
In [ ]: models_reg
```

```
Out[ ]: ConfigurableRegistry
                attr_name: _register_name
                unique: False
                Number of registered classes: 11
                Registered classes: ['unet', 'unet++', 'manet', 'linknet', 'fpn', 'psp', 'pan', 'deeplabv3', 'deeplabv3plus', 'Net', 'Ne
        tBig']
                Class key: model
                Attribute args: _default_args
                Attribute kwargs: _default_kwargs
                Additional info from attributes: []
```

```
In [ ]: model_args = {}
        model = models_reg.get_instance('Net', **model_args)
        model
```

```
Out[ ]: NeuralNetworkDecorated(
          (flatten): Flatten(start_dim=1, end_dim=-1)
          (linear_relu_stack): Sequential(
            (0): Linear(in_features=784, out_features=512, bias=True)
            (1): ReLU()
            (2): Linear(in_features=512, out_features=512, bias=True)
            (3): ReLU()
            (4): Linear(in_features=512, out_features=10, bias=True)
          )
        )
```

Backbones registry - Integration with smp

```
In [ ]: import typing

        import torch
        import torch.nn as nn
        import segmentation_models_pytorch as smp

        from segwork.model import backbones_reg
```

```
In [ ]: backbones_reg
```

Out[ ]:
```
ConfigurableRegistry
        attr_name: _register_name
        unique: False
        Number of registered classes: 113
        Registered classes: ['resnet18', 'resnet34', 'resnet50', 'resnet101', 'resnet152', 'resnext50_32x4d', 'resnext101_32x4
d', 'resnext101_32x8d', 'resnext101_32x16d', 'resnext101_32x32d', 'resnext101_32x48d', 'dpn68', 'dpn68b', 'dpn92', 'dpn98', 'dpn
107', 'dpn131', 'vgg11', 'vgg11_bn', 'vgg13', 'vgg13_bn', 'vgg16', 'vgg16_bn', 'vgg19', 'vgg19_bn', 'senet154', 'se_resnet50',
'se_resnet101', 'se_resnet152', 'se_resnext50_32x4d', 'se_resnext101_32x4d', 'densenet121', 'densenet169', 'densenet201', 'dense
net161', 'inceptionresnetv2', 'inceptionv4', 'efficientnet-b0', 'efficientnet-b1', 'efficientnet-b2', 'efficientnet-b3', 'effici
entnet-b4', 'efficientnet-b5', 'efficientnet-b6', 'efficientnet-b7', 'mobilenet_v2', 'xception', 'timm-efficientnet-b0', 'timm-e
fficientnet-b1', 'timm-efficientnet-b2', 'timm-efficientnet-b3', 'timm-efficientnet-b4', 'timm-efficientnet-b5', 'timm-efficient
net-b6', 'timm-efficientnet-b7', 'timm-efficientnet-b8', 'timm-efficientnet-l2', 'timm-tf_efficientnet_lite0', 'timm-tf_efficien
tnet_lite1', 'timm-tf_efficientnet_lite2', 'timm-tf_efficientnet_lite3', 'timm-tf_efficientnet_lite4', 'timm-resnest14d', 'timm-
resnest26d', 'timm-resnest50d', 'timm-resnest101e', 'timm-resnest200e', 'timm-resnest269e', 'timm-resnest50d_4s2x40d', 'timm-res
nest50d_1s4x24d', 'timm-res2net50_26w_4s', 'timm-res2net101_26w_4s', 'timm-res2net50_26w_6s', 'timm-res2net50_26w_8s', 'timm-res
2net50_48w_2s', 'timm-res2net50_14w_8s', 'timm-res2next50', 'timm-regnetx_002', 'timm-regnetx_004', 'timm-regnetx_006', 'timm-re
gnetx_008', 'timm-regnetx_016', 'timm-regnetx_032', 'timm-regnetx_040', 'timm-regnetx_064', 'timm-regnetx_080', 'timm-regnetx_12
0', 'timm-regnetx_160', 'timm-regnetx_320', 'timm-regnety_002', 'timm-regnety_004', 'timm-regnety_006', 'timm-regnety_008', 'tim
m-regnety_016', 'timm-regnety_032', 'timm-regnety_040', 'timm-regnety_064', 'timm-regnety_080', 'timm-regnety_120', 'timm-regnet
y_160', 'timm-regnety_320', 'timm-skresnet18', 'timm-skresnet34', 'timm-skresnext50_32x4d', 'timm-mobilenetv3_large_075', 'timm-
mobilenetv3_large_100', 'timm-mobilenetv3_large_minimal_100', 'timm-mobilenetv3_small_075', 'timm-mobilenetv3_small_100', 'timm-
mobilenetv3_small_minimal_100', 'timm-gernet_s', 'timm-gernet_m', 'timm-gernet_l']
        Class key: encoder
        Attribute args: _default_args
        Attribute kwargs: params
        Additional info from attributes: ['pretrained_settings']
```

In [ ]:
```python
backbones_reg.add_additional_args('_description')

@backbones_reg.register
class DummyBackboneDecorated(nn.Module, smp.encoders._base.EncoderMixin):
    """Dummyy encoder to test compatibility with smp architectures

    Testing:
     - Custom attributes in registry
      - To be used in smp framework it is regquired to inherit from EncoderMixin
    """

    _register_name='Net'

    # Default params
    params = {
        'out_channels' : (3, 64, 256, 512),
        'depth': 3
    }

    # Additional settings
    pretrained_settings = None

    _description = 'Formal description of encoder'

    def __init__(self, out_channels: typing.List, depth:int):
        super(DummyBackboneDecorated, self).__init__()

        # A number of channels for each encoder feature tensor, list of integers
        self._out_channels: typing.Iterable[int] = out_channels

        # A number of stages in decoder (in other words number of downsampling operations), integer
        # use in in forward pass to reduce number of returning features
        self._depth: int = depth

        # Default number of input channels in first Conv2d layer for encoder (usually 3)
        self._in_channels: int = 3

        blocks = []

        for idx in range(len(out_channels) - 1):
            blocks.append(nn.Sequential(
            nn.Conv2d(out_channels[idx], out_channels[idx + 1], 3, padding=1),
            nn.Conv2d(out_channels[idx + 1], out_channels[idx + 1], 3, stride=2, padding=1),
        ))

        self.stages = nn.Sequential(*blocks)

    def forward(self, x):
        out = [x]

        for stage in self.stages:
            x = stage(x)
            out.append(x)

        return out
```

In [ ]:
```python
encoder_name = 'Net'

# Framework entrypoint
backbone_fr = backbones_reg.get_instance(encoder_name)
```

```python
# SMP entrypoint compatibility
backbone = smp.encoders.get_encoder(encoder_name)

# print(backbone)
print(list(backbones_reg['Net'].keys()))
print(list(backbones_reg['resnet34'].keys()))
```

```
['encoder', '_default_args', 'params', 'pretrained_settings', '_description']
['encoder', 'pretrained_settings', 'params']
```

Output of registered backbone

In [ ]:

```python
x = torch.rand(1,3,224,224)

out = (backbone(x))

print('Features size...')
for idx, f in enumerate(out):
    print(f'Stage {idx:02d}: {f.size()}')
```

```
Features size...
Stage 00: torch.Size([1, 3, 224, 224])
Stage 01: torch.Size([1, 64, 112, 112])
Stage 02: torch.Size([1, 256, 56, 56])
Stage 03: torch.Size([1, 512, 28, 28])
```

Using custom bakcbone

In [ ]:

```python
model_args = {
    'encoder_name' : 'Net',
    'encoder_depth' : 3,
    'encoder_weights' : None,
    'decoder_channels' : (512, 256, 64),
    'in_channels' : 3,
    'classes' : 20
}

model = smp.Unet(**model_args)
```

In [ ]:

```python
out = model(x)
out.size()
```

Out[ ]:
```
torch.Size([1, 20, 224, 224])
```

In [ ]:
```
model_fr = models_reg.get_instance('unet', **model_args)
```

In [ ]:
```
out_fr = model_fr(x)
out_fr.size()
```

Out[ ]:
```
torch.Size([1, 20, 224, 224])
```