intel

OpenVINO™

CVPR 2024

Edge-Optimized Deep Learning: Harnessing Generative AI and Computer Vision with Open-Source Libraries

Module 1

Data Management, Training, and Fine-tuning Computer Vision Tasks

Samet Akcay

Intel NEX SW

Contributors: Harim Kang, Jaeguk Hyun

Agenda

By the end of this session, you will learn how to

01

Dataset

Zero-shot to annotate

Automated annotation

Datumaro dataset management.

02

Train / Test

Zero-shot SAM
Classification
Object Detection
Instance Segmentation

03

Export

What are the BKMs to run this model?

04

XAI

Create explainability saliency maps with the IR model

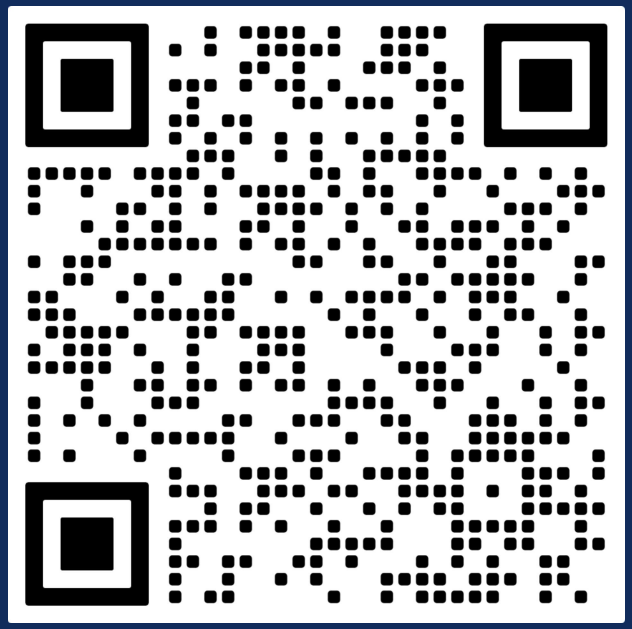
05

Deploy / Demo

Create a demo package ready to be deployed

OTX

Introduction



What is OpenVINO Training eXtensions – OTX?

- One-stop shop of verified algorithms for many vision tasks and learning methods
- Provides simple CLI and API for quick start without hassles
- Full OpenVINO integration for model optimization, inference and deployment.

What is OpenVINO Training eXtensions – OTX?

One command is all you need

CLI

```
# <train, test, export, explain, deploy>
# $ otx <entrypoint> --arg value
$ otx train \
   --task detection  \
   --data_root /path/to/data …
```

API

```
>>> from otx.engine import Engine
>>> engine = Engine(data_root="data/wgisd")
>>> engine.train()
```

What is OpenVINO Training eXtensions – OTX?

01

Dataset

How does my data look?

02

Models

Which AI model can I use?

03

BKM

What are the BKMs to run this model?

04

Improve

Is this the highest performance I can achieve?

05

Optimize

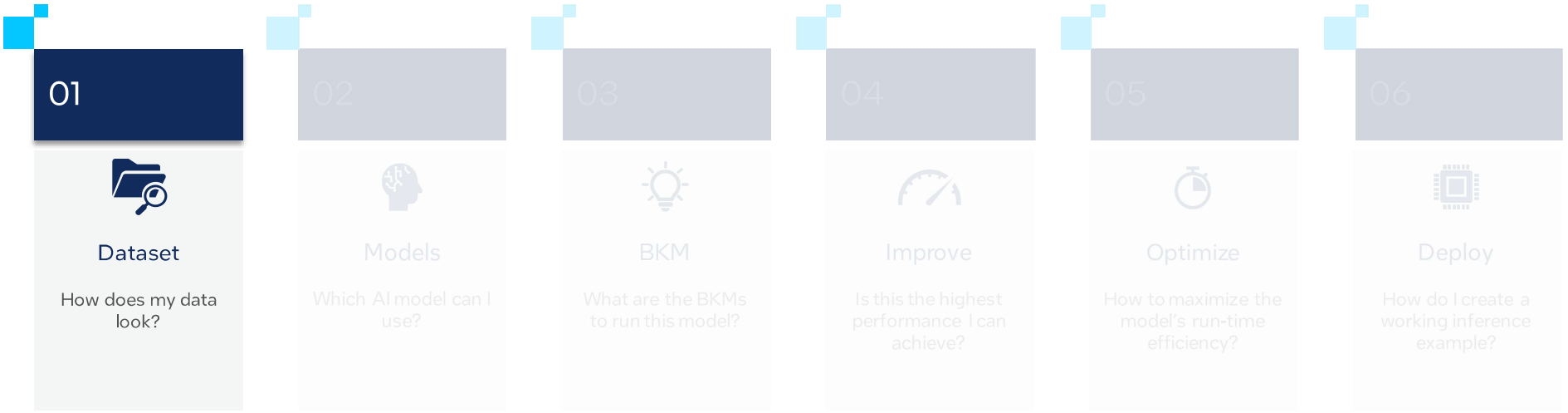
How to maximize the model's run-time efficiency?

06

Deploy

How do I create a working inference example?

What is OpenVINO Training eXtensions – OTX?



01 – Leverage Datumaro, the data frontend of OTX

What is OpenVINO Training eXtensions – OTX?

01 Dataset
How does my data look like

02 Models
Which AI model can I use?

03 BKM
What are the BKMs to run this model?

04 Improve
Is this the highest performance I can achieve?

05 Optimize
How to maximize the model's run-time efficiency?

06 Deploy
How do I create a working inference example?

02 – Use OpenVINO-verified models for best performance and efficiency

```
# CLI
$ otx find
```

```
# API
>>> from otx.models import list_models
>>> list_models(task="DETECTION")
```

What is OpenVINO Training eXtensions – OTX?

01	02	03	04	05	06
Dataset	Models	BKM	Improve	Optimize	Deploy
How does my data look like	Which AI model can I use?	What are the BKMs to run this model?	Is this the highest performance I can achieve?	How to maximize the model's run-time efficiency?	How do I create a working inference example?

03/04 – Autoconfiguration will find the best task type, model and parameters. Advanced customization is also possible

```
# CLI
$ otx train
```

```
# API
>>> from otx.engine import Engine
>>> engine = Engine(data_root="data/wgisd")
>>> engine.train()
```

What is OpenVINO Training eXtensions – OTX?

01	02	03	04	05	06
Dataset	Models	BKM	Improve	Optimize	Deploy
How does my data look like	Which AI model can I use?	What are the BKMs to run this model?	Is this the highest performance I can achieve?	How to maximize the model's run-time efficiency?	How do I create a working inference example?

05 – Use OpenVINO NNCF-based model optimization and export runnable codes with model

```
# CLI
$ otx export
$ otx optimize
```

```
# API
>>> ir_model_path = engine.export()
>>> engine.optimize(ir_model_path)
```

What is OpenVINO Training eXtensions – OTX?

01	02	03	04	05	06
Dataset	Models	BKM	Improve	Optimize	Deploy
How does my data look like	Which AI model can I use?	What are the BKMs to run this model?	Is this the highest performance I can achieve?	How to maximize the model's run-time efficiency?	How do I create a working inference example?

06 –Evaluate, explain and deploy models with built-in CLI and API

```
# CLI
$ otx explain
$ otx export
$ python demo.py
```

```
# API
>>> engine.explain(checkpoint="<checkpoint-path>",
datamodule=OTXDataModule(...),explain_config=ExplainConfig(postprocess=True ),dump=True )
>>> engine.export(export_format='EXPORTABLE_CODE')
```

OTX

Features

Installation

Lightweight, hardware-agnostic installation

CLI

```
# <train, test, export, explain, deploy>
# Install via PyPI
$ pip install otx

# Multiple installation options
otx install --help
```

Features

End-to-end DL pipeline for all levels – *From beginner to Advanced*

Task Types

- Classification
- Detection, Rotated Detection
- Semantic and Instance Segmentation
- Anomaly Detection
- Action Recognition
- Visual Prompting

Learning Methods

- Fully-supervised
- Semi-supervised
- Self-supervised
- Class Incremental
- Imbalanced

API / CLI Functionality

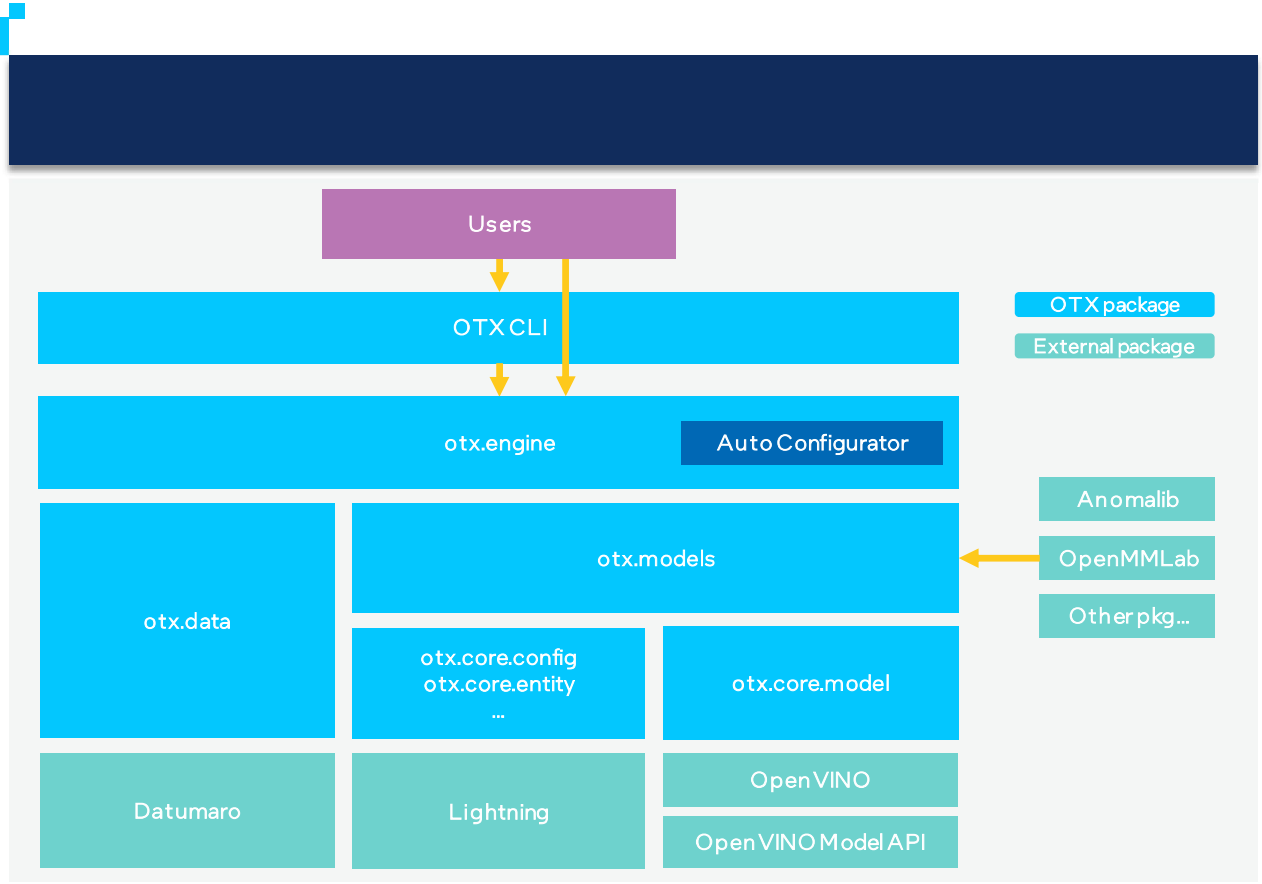
- Auto-installation
- Auto-learning method
- Integrated Image Tiling
- Hyper-parameter Optimization
- OpenVINO Optimization
- Integrated Explainable AI (XAI)

OTX Architecture

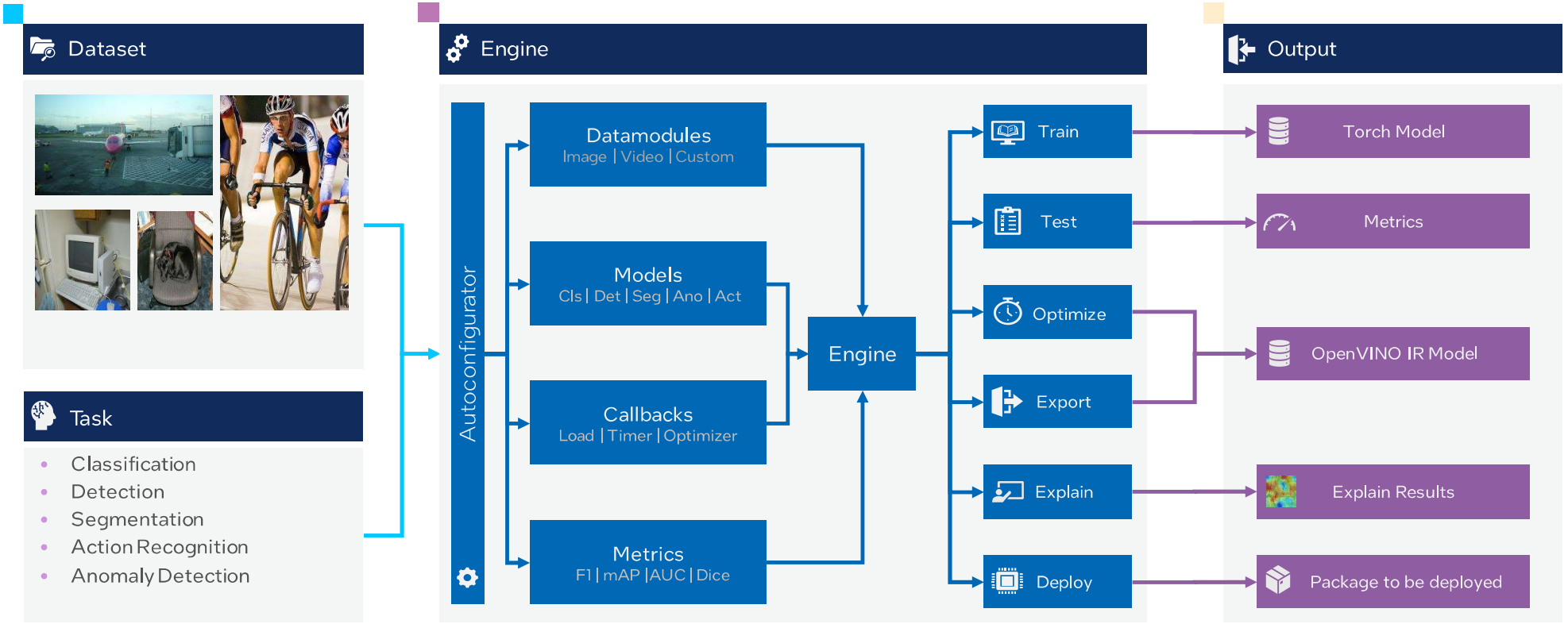
OTX Architecture

API/CLI Parity

- Jsonargparse-based CLI
- Lightning-based Data and Model
- Engine orchestrates the pipeline.



Autoconfiguration



Autoconfiguration

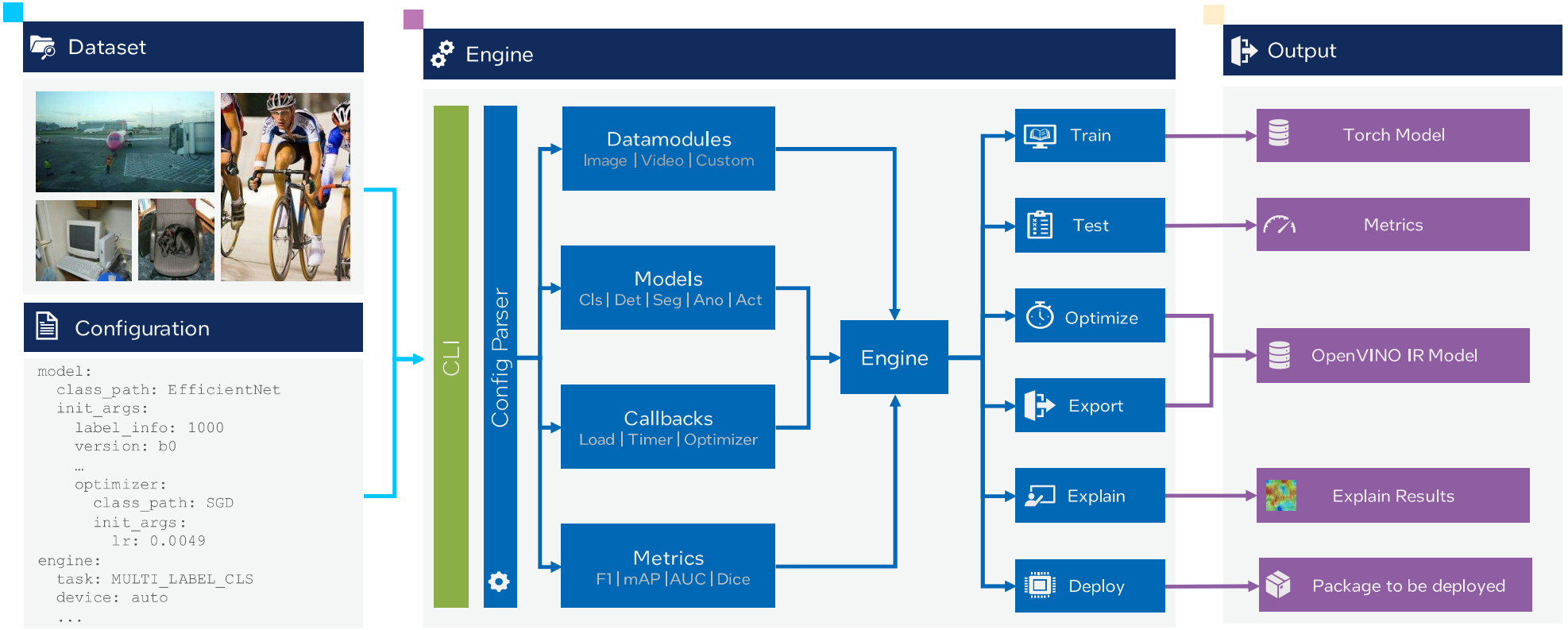
CLI

```
# TASKS:
# - MULTI_LABEL_CLS, MULTI_CLASS_CLS, DETECTION,
# - INSTANCE_SEGMENTATION, SEMANTIC_SEGMENTATION,
# - ACTION_RECOGNITION
$ otx train \
    --task <TASK> \
    --data_root data/VOCdevkit/VOC2012 \
    --data.config.data_format voc
```

API

```
# API via config
from otx.engine import Engine

engine = Engine(
    data_root=data_root,
    task="INSTANCE_SEGMENTATION",
    work_dir="otx-workspace-api-ins-seg-auto",
)

engine.train(max_epochs=3)
```

via Config File



End-to-End Training via Config File

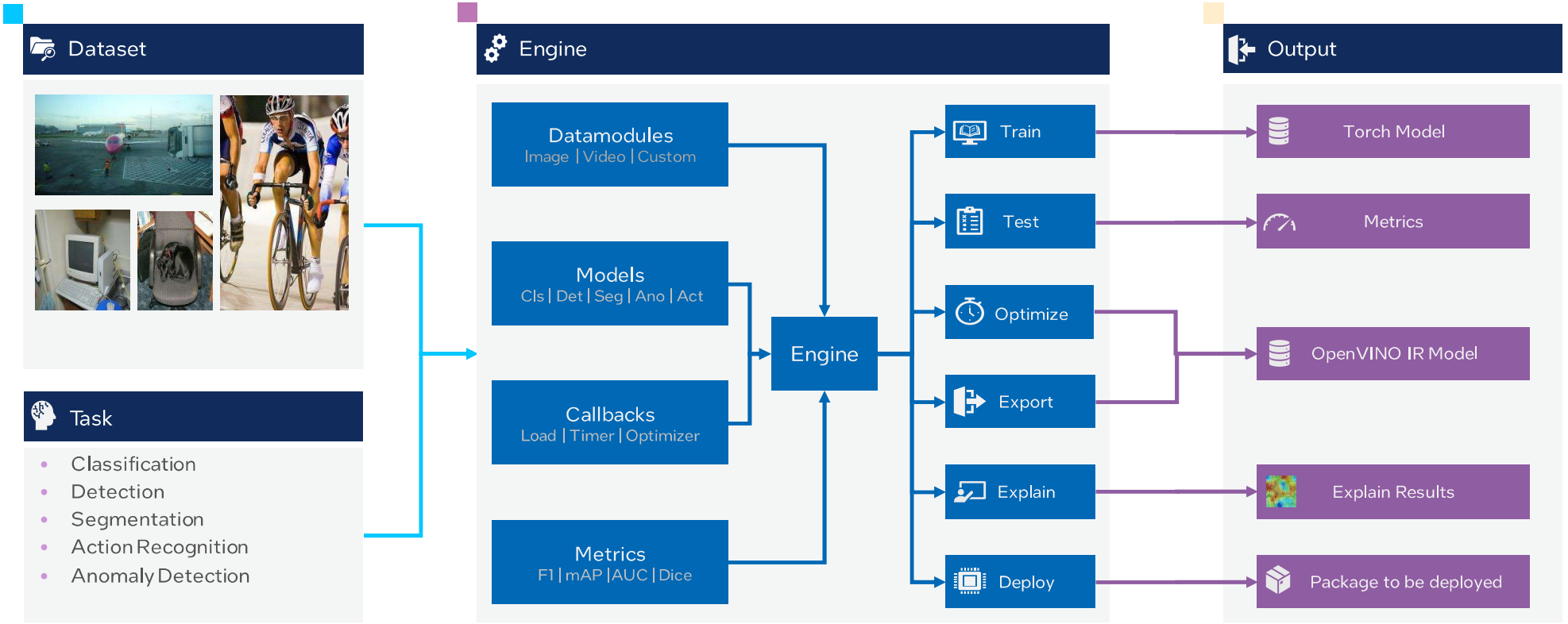
CLI

```
# Train via config file
$ otx train \
    --config efficientnet_b0_light.yaml \
    # Overwrite some arguments (Optional)
    --data_root data/VOCdevkit/VOC2012 \
    --data.config.data_format voc \
    --work_dir otx-workspace-api-multi-label-cls
```

API

```
# API via config
from otx.engine import Engine

data_root = "data/VOCdevkit/VOC2012"
recipe = "src/otx/recipe/classification/multi_label_cls/
            "efficientnet_b0_light.yaml"
override_kwargs = {"data.config.data_format": "voc"}
engine = Engine.from_config(
  config_path=recipe,
  data_root=data_root,
  work_dir="otx-workspace-api-multi-label-cls",
  **override_kwargs,
)

engine.train(max_epochs=2, precision=16)
```

via API Modules



via API Modules

CLI

```
# CLI via API modules
$ otx train
    --config src/otx/recipe/classification/
          multi_label_cls/efficientnet_b0_light.yaml \
    --data_root data/VOCdevkit/VOC2012 \
    --data.config.data_format voc \
```

API

```
# API via modules
>>> datamodule = OTXDataModule(
...     task="MULTI_LABEL_CLS",
...     config=DataModuleConfig(
...         data_format="voc",
...         data_root=data_root,
...           train_subset=SubsetConfig(
...           subset_name="train",
...           batch_size=8,
...           num_workers=2,
...           transform_lib_type="MMPRETRAIN",
...           transforms=simple_transforms,
...         val_subset=SubsetConfig(...),
...         test_subset=SubsetConfig(...),
...     ),
... ),

>>> model = EfficientNetB0ForMultilabelCls(num_classes)

>>> engine = Engine(datamodule=datamodule,model=model)
>>> engine.train(max_epochs=2)
```

OTX

Additional Features

Additional OTX Features

01

Custom Data and Models

It is possible to create custom data and models.

02

Image Tiling

Improve performance for small object detection

03

XPU Support

Train/Test models using Intel XPUs

04

HPO

Hyper-parameter optimization to tune the model performance

05

Export to Different Precision

Possibility to export to different precision such as int8

06

Optimization

Optimization support including post-training quantization

Custom Data

API

```
# Add Image Tiling API Example Here.
my_transforms = [
   Resize(size=[224, 224]),
   # Your list of custom transforms here.
]

datamodule = OTXDataModule(task="MULTI_LABEL_CLS",
   config=DataModuleConfig(
      data_format="voc",
      data_root=data_root,
      train_subset=SubsetConfig(
           subset_name="train",
           batch_size=32,
           num_workers=2,
           transform_lib_type="TORCHVISION",
           transforms= my_transforms,
       ),
       val_subset=SubsetConfig(...),
       test_subset=SubsetConfig(...
       ),
       ),
)
```

Custom Model

Create torchvision models

CLI

```
otx train \
    --model otx.algo.classification.OTXTVModel \
    --model.backbone convnext_small \
    --data_root otx_v2_dataset/multiclass_CUB_small/1 \
    --work_dir otx-workspace-convnext \
    --max_epochs 2
```

API

```
# Imports
from otx.algo.classification import OTXTVModel
from otx.engine import Engine

# Create a torchvision model
tv_model = OTXTVModel(backbone="convnext_small", label_info=2)

# Multi-Class Classification
engine = Engine(
   data_root="otx_v2_dataset/multiclass_CUB_small/1",
   model=tv_model,
   work_dir="otx-workspace-tv-model",
)
engine.train(max_epochs=2)
```

Custom Model

Custom objective functions

API

```
from otx.algo.classification.efficientnet import EfficientNetForMultilabelCls
from otx.algo.classification.losses. import AsymmetricAngularLossWithIgnore

model = EfficientNetForMultilabelCls(
   label_info=datamodule.label_info,
   loss_callable=AsymmetricAngularLossWithIgnore(),
)

# Multi-Label Classification
engine = Engine(
   datamodule=datamodule,
   model=model,
   work_dir="otx-workspace-api-multi-label-cls",
)

engine.train(max_epochs=2)
```

Image Tiling

CLI

```
# Add Image Tiling CLI Example Here.
$ otx train
     ...
     --data.config.tile_config.enable_tiler True
```

API

```
# Add Image Tiling API Example Here.
>>> datamodule = OTXDataModule(
...     task="DETECTION",
...     config=DataModuleConfig(
...             …
...             tile_config=TileConfig(enable_tiler=True),
...      ),

... ),

>>> engine = Engine(datamodule=datamodule,model=model)
>>> engine.train(max_epochs=2)
```

XPU Support

CLI

```
# CLI - Installation for XPU support
$ pip install '.[xpu]'
>     --extra-index-url https://pytorch-extension.intel.com/release-whl/stable/xpu/us/
>    --data.config.tile_config.enable_tiler True

$ source /path/to/intel/oneapi/setvars.sh
$ export LD_PRELOAD=/usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.30
$ export IPEX_FP32_MATH_MODE=TF32

# CLI - XPU Support
$ otx train
    ...
    --engine.device xpu
```

API

```
# API - XPU Support
>>> from otx.engine import Engine
>>> engine = Engine(…, device='xpu')
>>> engine.train()
```

Hyper-parameter Optimization

CLI

```
# CLI - HPO
$ otx train
     ...
     --run_hpo True
```

API

```
# API - HPO
>>> from otx.engine import Engine
>>> engine = Engine(..)
>>> engine.train(run_hpo=True)
```

Export to Different Precision

CLI

```
# CLI - Precision Example
$ otx export
     ...
     --export_precision FP16
```

API

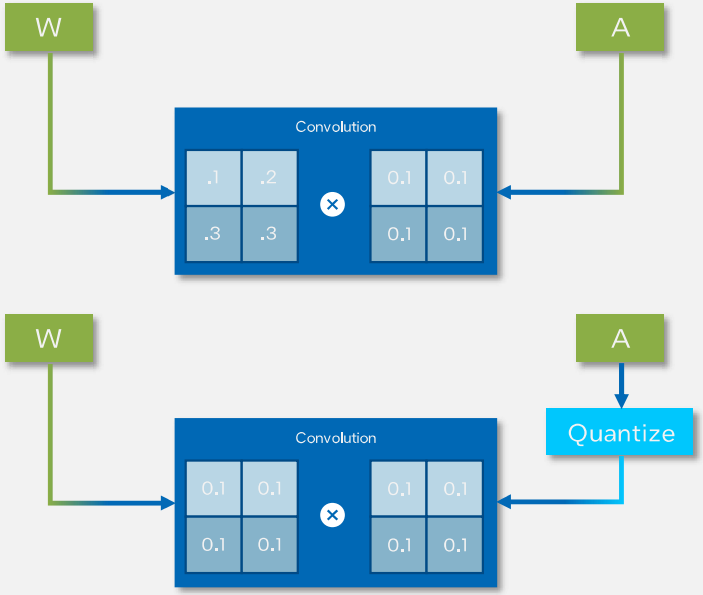
```
# API - Precision Example
>>> from otx.engine import Engine
>>> engine = Engine(..)
>>> engine.export(export_precision='FP16')
```

Optimization via NNCF

API

```python
import nncf
import openvino.runtime as ov
from torch.utils.data import DataLoader

from torchvision.datasets import ImageFolder
from torchvision.transforms import Compose, ToTensor
from nncf.Dataset import NNCFDataset

# Instantiate your uncompressed model
model = ov.Core().read_model("/path/to/model.xml")

# Provide validation part of the dataset to collect statistics needed
# for the compression algorithm
transforms = Compose([ToTensor()])
val_dataset = ImageFolder("/path/to/dataset", transform=transforms)
val_dataloader = DataLoader(val_dataset, batch_size=1, shuffle=False)

# Step 1: Initialize the transform function
def transform_fn(data_item):
images, _ = data_item
return images

# Step 2: Initialize the NNCF dataset
calibration_dataset = NNCFDataset(val_dataloader, transform_fn)

# Step 3: Runt the quantization pipeline
quantized_model = nncf.quantize(model, calibration_dataset)
```

High-Level Diagram



Practical Implementation

Get Started

Installation

Installation

PyPI Install

```
python -m venv .otx
source .otx/bin/activate

# Install OTX CLI
pip install otx
# Install the full functionality via OTX CLI
otx install -v
```

Notebook

Documentation

https://github.com/openvinotoolkit/training_extensions/blob/tutorials/cvpr24/notebooks/000_install.ipynb

Installation

Source Installation

```
git clone https://github.com/openvinotoolkit/training_extensions.git
cd training_extensions

python -m venv .otx && source .otx/bin/activate

pip install -e .
otx install -v
```

Notebook

Documentation

https://github.com/openvinotoolkit/training_extensions/blob/tutorials/cvpr24/notebooks/000_install.ipynb

Use Case

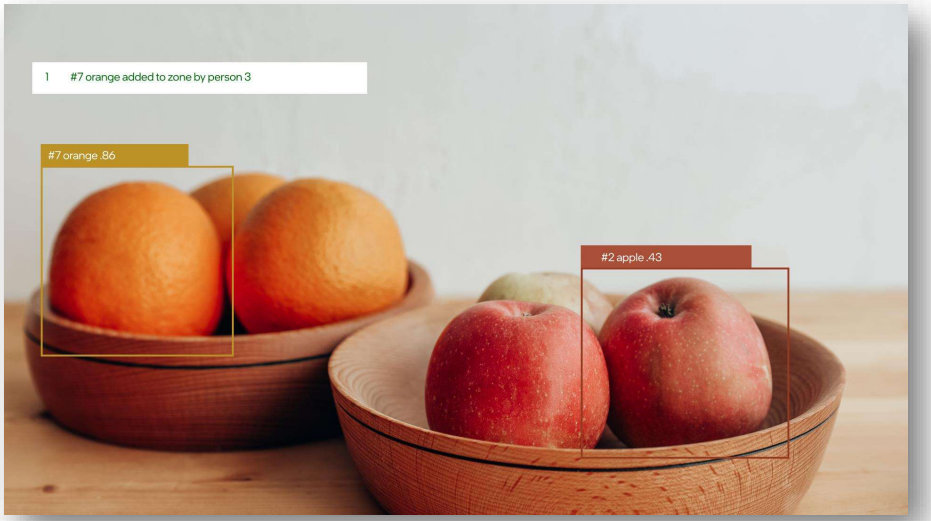
Problem Definition

Self Checkout Dataset – *Representative example*

Classification | Detection | Segmentation | VLMs

Self-Checkout in Retail: Challenges

- Real-time scalability
- Model performance
- Memory-efficiency and low-power to run on edge



https://github.com/openvinotoolkit/training_extensions/blob/tutorials/cvpr24/notebooks/000_install.ipynb

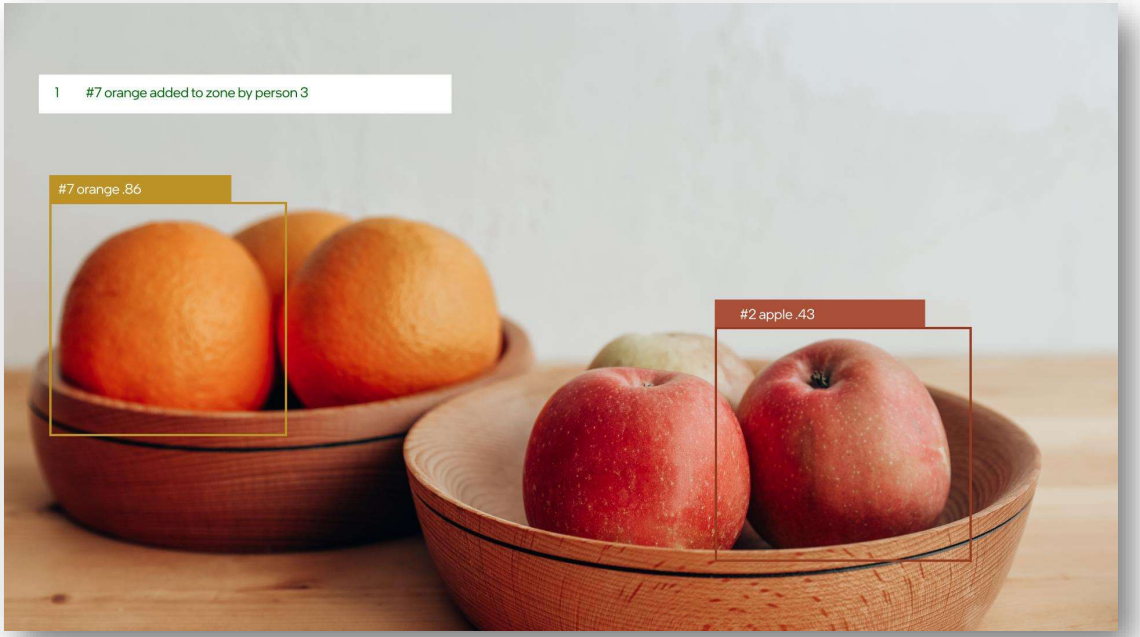
Self-Checkout in Retail: Solutions

01 Zero-shot Visual Prompting

02 Classification

03 Detection

04 Segmentation

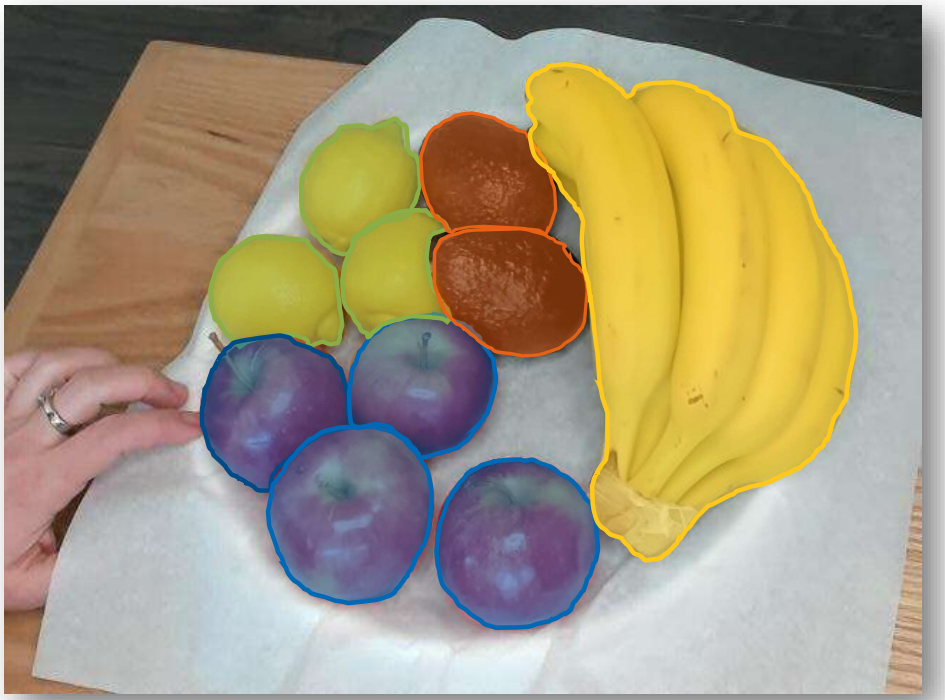


https://github.com/openvinotoolkit/training_extensions/blob/tutorials/cvpr24/notebooks/000_install.ipynb

Task Types

Zero-Shot Visual Prompting

Zero-shot via SAM



```
>>> datamodule = OTXDataModule(
...     task="MULTI_LABEL_CLS",
...     config=DataModuleConfig(
...         data_format="voc",
...         data_root=data_root,
...         train_subset=SubsetConfig(
...         subset_name="train",
...         batch_size=8,
...         num_workers=2,
...         transform_lib_type="MMPRETRAIN",
...         transforms=simple_transforms,
...      ),
...      val_subset=SubsetConfig(...),
...      test_subset=SubsetConfig(...),
... ),

>>> model = EfficientNetB0ForMultilabelCls(num_classes)

>>> engine = Engine(datamodule=datamodule,model=model)
>>> engine.train(max_epochs=2)
```

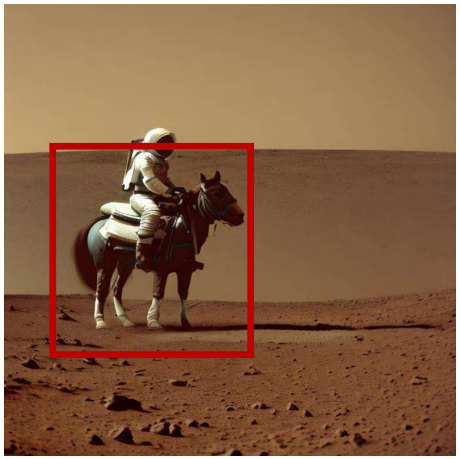
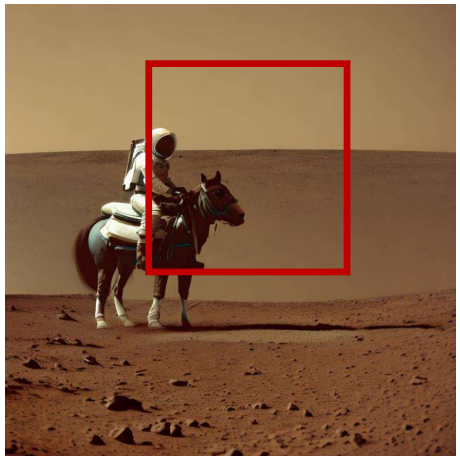
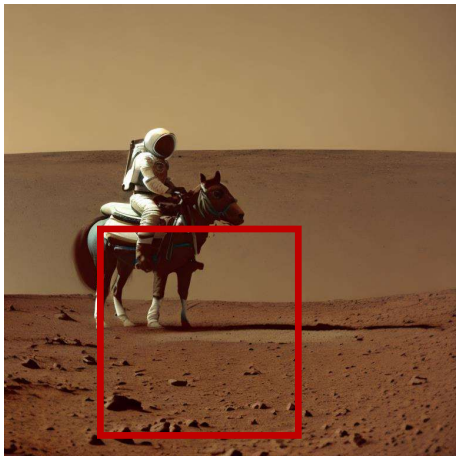
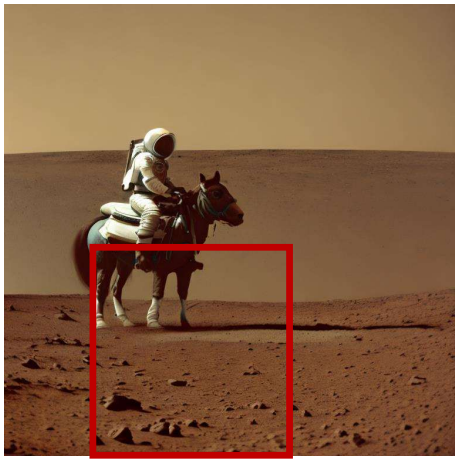
https://github.com/openvinotoolkit/training_extensions/blob/tutorials/cvpr24/notebooks/001_zero_shot_visual_prompting.ipynb

Zero-shot via SAM

```python
from otx.core.config.data import DataModuleConfig, SubsetConfig
from otx.core.data.module import OTXDataModule
from otx.algo.classification.efficientnet_b0 import EfficientNetB0ForMultilabelCls
from otx.engine import Engine


simple_transforms = [
    {"type": "LoadImageFromFile"},
    {"type": "Resize", "scale": 224, "backend": "cv2"},
    {"type": "PackInputs"},
]

datamodule = OTXDataModule(
    task="MULTI_LABEL_CLS",
    config=DataModuleConfig(
        data_format="voc",
        data_root=data_root,
        train_subset=SubsetConfig(
            subset_name="train",
            batch_size=8,
            num_workers=2,
            transform_lib_type="MMPRETRAIN",
            transforms=simple_transforms,
        ),
        val_subset=SubsetConfig(
            subset_name="val",
            batch_size=8,
            num_workers=2,
            transform_lib_type="MMPRETRAIN",
            transforms=simple_transforms,
        ),
        test_subset=SubsetConfig(
            subset_name="val",
            batch_size=8,
            num_workers=2,
            transform_lib_type="MMPRETRAIN",
            transforms=simple_transforms,
        ),
    ),
)

model = EfficientNetB0ForMultilabelCls(num_classes=datamodule.label_info.num_classes)

engine = Engine(
    datamodule=datamodule,
    model=model,
    work_dir="otx-workspace-api-multi-label-cls",
)

engine.train(max_epochs=2)
```

Zero-shot via SAM

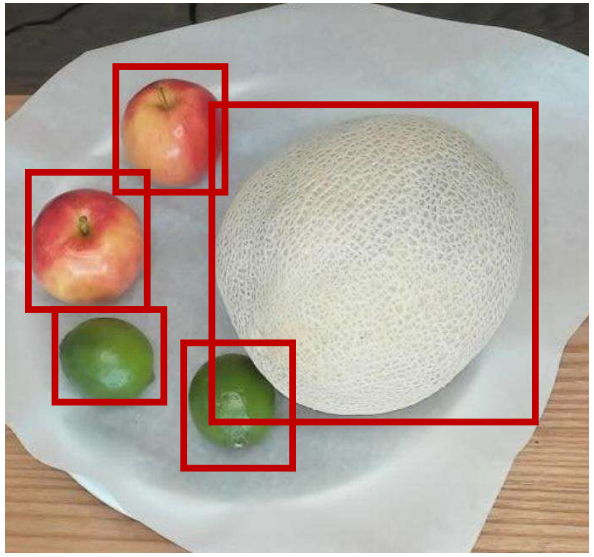
Results – Success/Failure

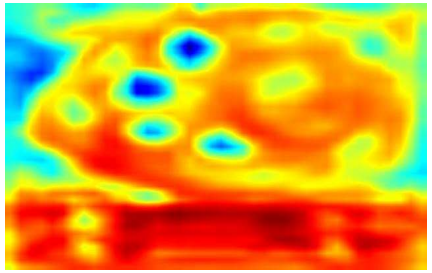





Task Types

Detection

Train, evaluate, export, and explain a det model in 6 lines of code





API Code

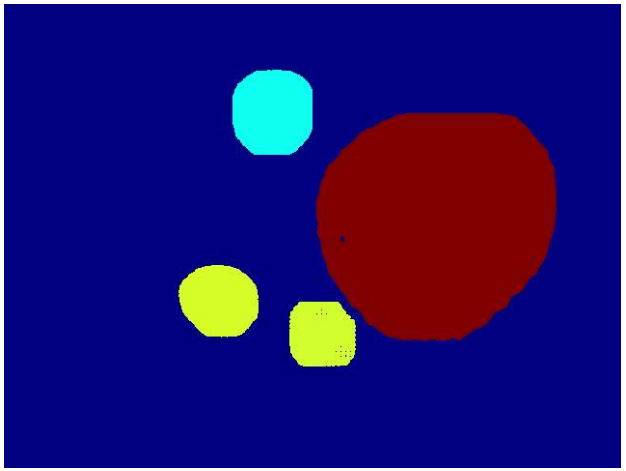
```
from otx.engine import Engine
from otx.core.config.explain import ExplainConfig

recipe = "otx/recipe/detection/yolox_s.yaml"
override_dataset_format = {"data.config.data_format": "datumaro"}
#Define model config
engine = Engine.from_config(config_path=recipe, data_root=data_root, work_dir=work_dir,
**override_dataset_format)
#Train model
engine.train(max_epochs=30)
#Test model
engine.test()
#Explain model
engine.explain(explain_config=ExplainConfig(postprocess=True), dump=True)
#Export model to OpenVINO IR Format
exported_ir_model_path = engine.export()
#Evaluate exported model
engine.test(checkpoint=exported_ir_model_path)
#Explain exported model
engine.explain(
    checkpoint=exported_ir_model_path,
    explain_config=ExplainConfig(postprocess=True),
    dump=True,
)
```

Before: 73.36% mAP
After Export: 50.07% mAP

Task Types

Segmentation

Build a Custom Instance Segmentation Model



API Code

```
from otx.engine import Engine
from otx.core.config.explain import ExplainConfig

recipe = "otx/recipe/instance_segmentation/maskrcnn_r50.yaml"
#Define model config
engine = Engine.from_config(config_path=recipe, data_root=data_root,
work_dir=work_dir)
#Train model
engine.train(max_epochs=30)
#Test model
engine.test()
#Explain model
engine.explain(explain_config=ExplainConfig(postprocess=True), dump=True)
#Export model to OpenVINO IR Format
exported_ir_model_path = engine.export()
#Evaluate exported model
engine.test(checkpoint=exported_ir_model_path)
#Explain exported model
engine.explain(
    checkpoint=exported_ir_model_path,
    explain_config=ExplainConfig(postprocess=True),
    dump=True,
)
```

Task Types

Classification

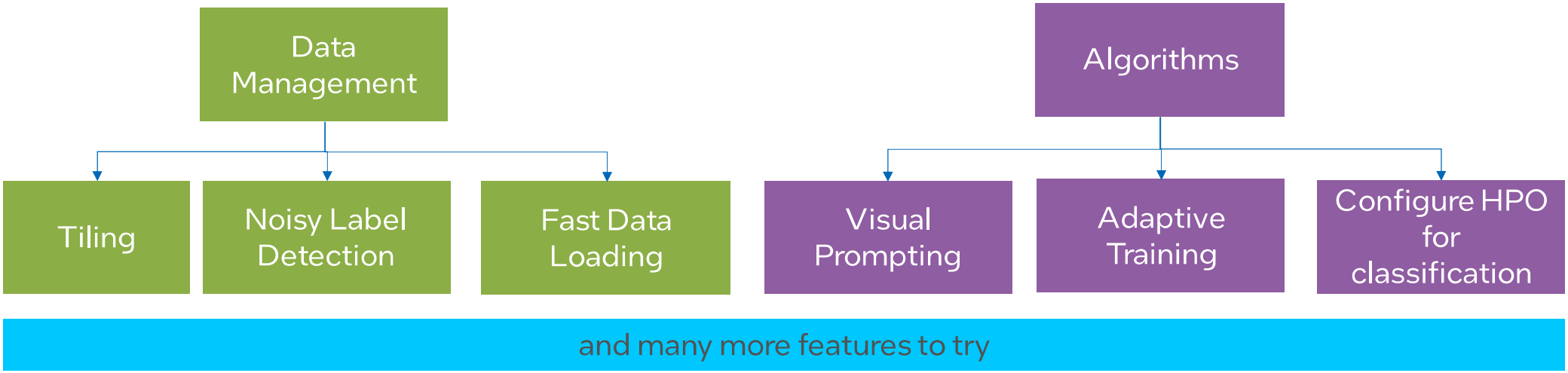
Build a Custom Multi-label Classification Model

API Code

```
from otx.engine import Engine
from otx.core.config.explain import ExplainConfig

recipe = "otx/recipe/classification/multi_label_cls/efficientnet_b0.yaml"
#Define model config
engine = Engine.from_config(config_path=recipe, data_root=data_root,
work_dir=work_dir)
#Train model
engine.train(max_epochs=30)
#Test model
engine.test()
#Explain model
engine.explain(explain_config=ExplainConfig(postprocess=True), dump=True)
#Export model to OpenVINO IR Format
exported_ir_model_path = engine.export()
#Evaluate exported model
engine.test(checkpoint=exported_ir_model_path)
#Explain exported model
engine.explain(
    checkpoint=exported_ir_model_path,
    explain_config=ExplainConfig(postprocess=True),
    dump=True,
)
```

Conclusion

Next steps?


Data Management
Tiling
Noisy Label Detection
Fast Data Loading
Algorithms
Visual Prompting
Adaptive Training
Configure HPO for classification
and many more features to try

https://github.com/openvinotoolkit/training_extensions