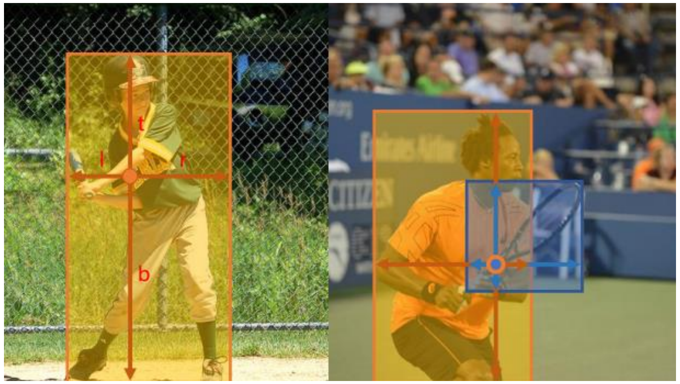
EE898 Programming Assignment 1

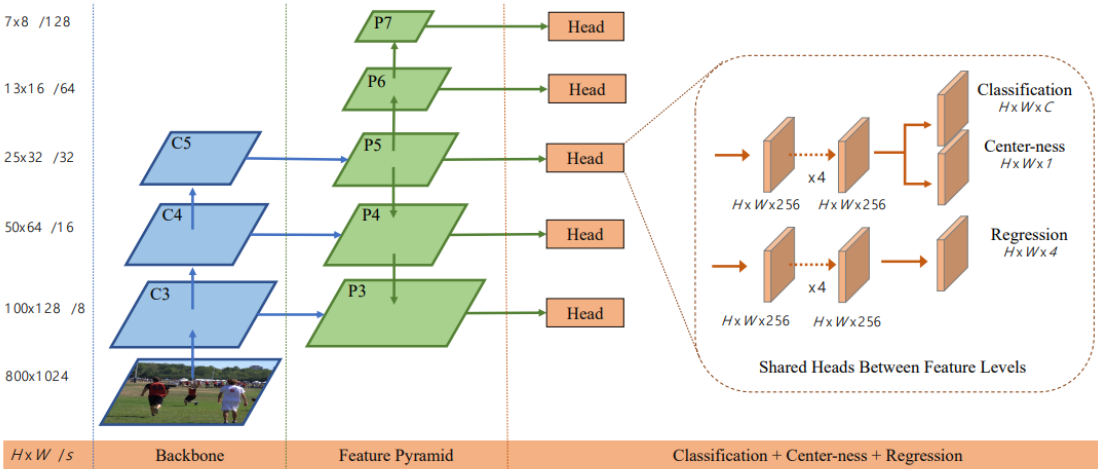
FCOS: Fully Convolutional One-Stage Object Detection

Youngtaek Oh (youngtaek.oh [at] kaist.ac.kr)

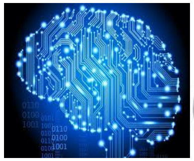
Programming Assignment 1

- In PA1, you will implement FCOS object detector, a fully convolutional anchor-free one-stage object detector.
- Please read the paper thoroughly to complete a baseline code and additional questions prepared in PA1.





FCOS: Fully Convolutional One-Stage Object Detection, ICCV 2019



Detectron2

https://github.com/facebookresearch/detectron2

Dependencies

- We provide detectron2-based skeleton code, where detectron2 provides many convenient tools such as data processing, train/test loop, dataset evaluation, and visualization of results.
- We hope all of you to pay attention solely on writing codes about detection pipelines and FCOS head itself.
- So, in order to train your implemented FCOS detector, you need to install detectron2 on your environment.
- We recommend installing detectron2 on Linux OS, Python>=3.6, Pytorch>=1.3.

Detectron2 Installation

- Requirements
  - Linux or macOS with Python ≥ 3.6
  - PyTorch ≥ 1.3
  - torchvision that matches the PyTorch installation.
  - OpenCV, optional, needed by demo and visualization
  - pycocotools: $ pip install cython; pip install -U 'git+https://github.com/cocodataset/cocoapi.git#subdirectory=PythonAPI'

- Build Detectron2 from source (recommended)

```
# install it from a local clone:
$ git clone https://github.com/facebookresearch/detectron2.git
$ cd detectron2 && python -m pip install -e .
```

https://github.com/facebookresearch/detectron2/blob/master/INSTALL.md

Common installation issues

- Common Installation Issues
  - Undefined torch/aten/caffe2 symbols, or segmentation fault immediately when running the library.
  - Undefined C++ symbols (e.g. `GLIBCXX`) or C++ symbols not found.
  - "Not compiled with GPU support" or "Detectron2 CUDA Compiler: not available".
  - "invalid device function" or "no kernel image is available for execution".
  - Undefined CUDA symbols or cannot open libcudart.so.
  - "ImportError: cannot import name '_C'".
  - ONNX conversion segfault after some "TraceWarning".

- See the downside of the webpage for solution.

https://github.com/facebookresearch/detectron2/blob/master/INSTALL.md

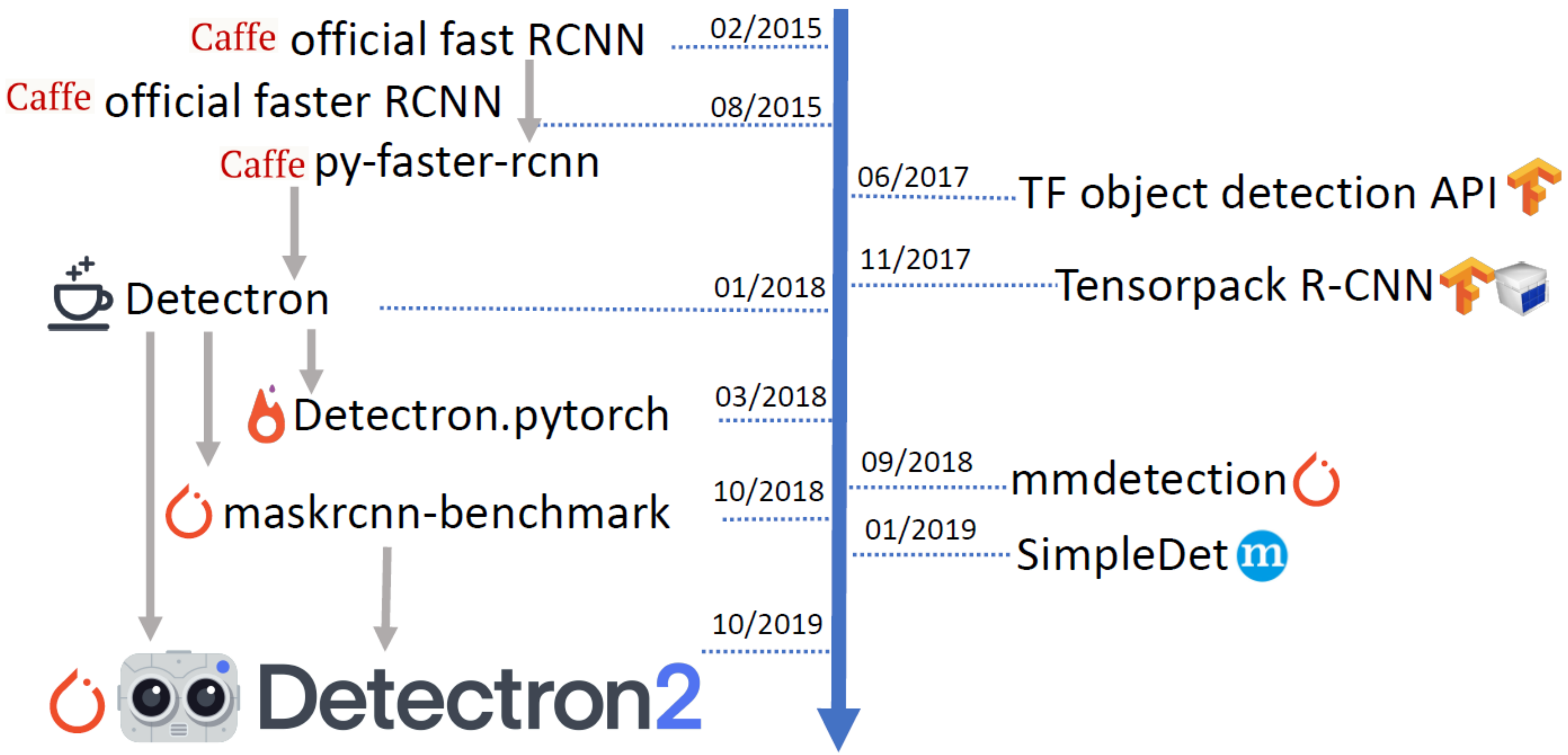
Detectron2 is …

- A library / research platform for object detection
    - Includes everything we just learned
    - And helps us build more

- In this tutorial:
    - What's in Detectron2
    - How to Use Detectron2 (beginners)
    - How to Use Detectron2 (advanced users)

slide by Yuxin Wu

Family of Detection/R-CNN Codebase



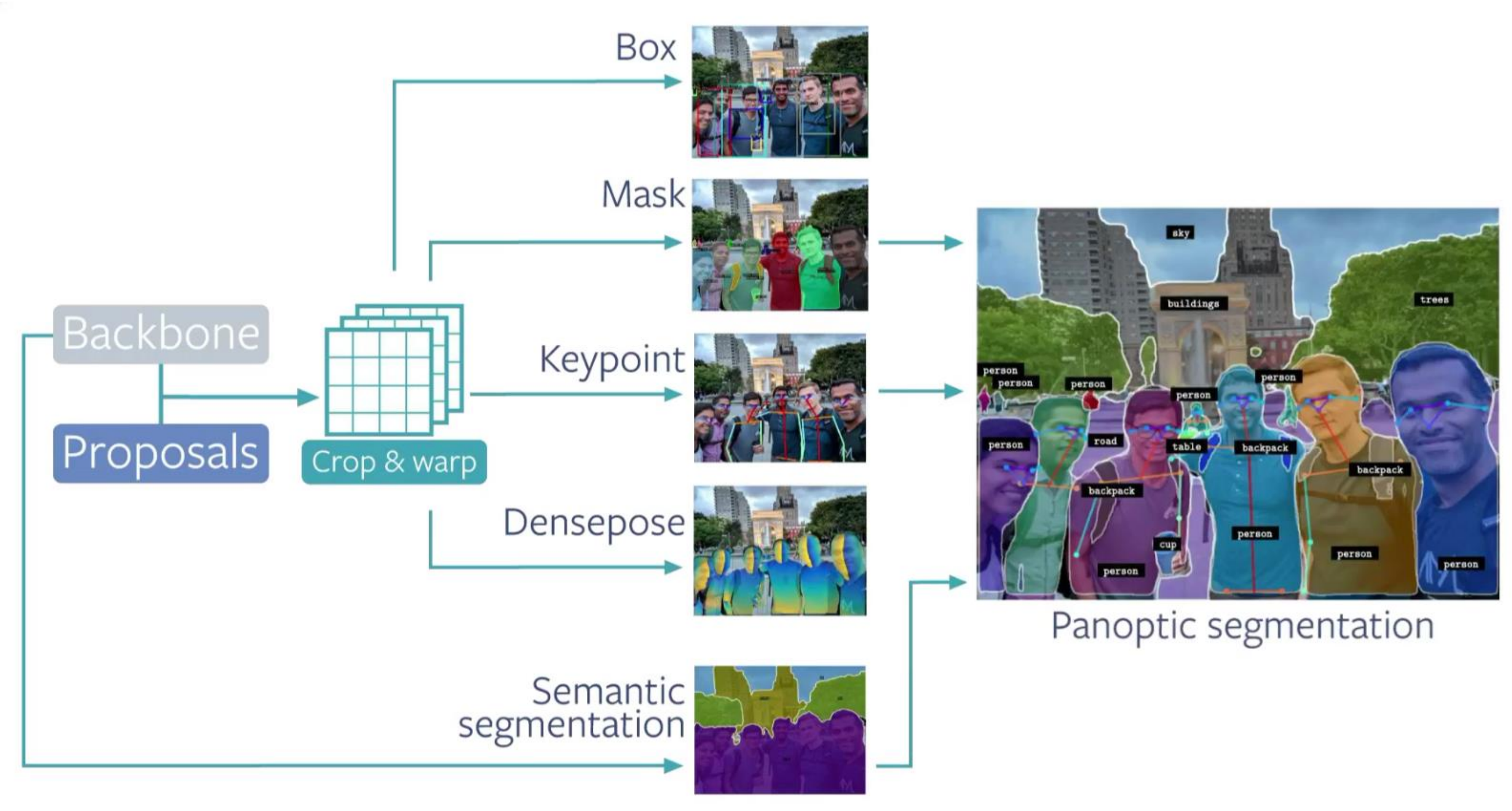
slide by Yuxin Wu

Why PyTorch (Eager Mode) for detection

- Tensors of irregular shapes
- Loops / control flow / heuristics
- More hackable for researcher

slide by Yuxin Wu

What's in Detectron2: Generalized R-CNN Models



slide by Yuxin Wu

What's in Detectron2

- Datasets:
    - COCO
    - LVIS
    - CityScapes
    - PascalVOC

- Tasks (data & evaluation):
    - (Rotated) Box
    - {Instance,Semantic,Panoptic} Segmentation
    - Person Keypoint, DensePose

slide by Yuxin Wu

What's in Detectron2: MODEL ZOO

- different models for users to play with.
- baselines for researchers.

Faster R-CNN:

Name	lr sched	train time (s/iter)	inference time (s/im)	train mem (GB)	box AP	model id	download
R50-C4	1x	0.593	0.110	4.8	35.7	137257644	model | metrics
R50-DC5	1x	0.380	0.068	5.0	37.3	137847829	model | metrics
R50-FPN	1x	0.210	0.055	3.0	37.9	137257794	model | metrics
R50-C4	3x	0.589	0.110	4.8	38.4	137849393	model | metrics
R50-DC5	3x	0.378	0.073	5.0	39.0	137849425	model | metrics
R50-FPN	3x	0.209	0.047	3.0	40.2	137849458	model | metrics
R101-C4	3x	0.656	0.149	5.9	41.1	138204752	model | metrics
R101-DC5	3x	0.452	0.082	6.1	40.6	138204841	model | metrics
R101-FPN	3x	0.286	0.063	4.1	42.0	137851257	model | metrics
X101-FPN	3x	0.638	0.120	6.7	43.0	139173657	model | metrics

RetinaNet:

Name	lr sched	train time (s/iter)	inference time (s/im)	train mem (GB)	box AP	model id	download
R50	1x	0.200	0.062	3.9	36.5	137593951	model | metrics
R50	3x	0.201	0.063	3.9	37.9	137849486	model | metrics
R101	3x	0.280	0.080	5.1	39.9	138363263	model | metrics

RPN & Fast R-CNN:

Name	lr sched	train time (s/iter)	inference time (s/im)	train mem (GB)	box AP	prop. AR	model id	download
RPN R50-C4	1x	0.130	0.051	1.5		51.6	137258005	model | metrics
RPN R50-FPN	1x	0.186	0.045	2.7		58.0	137258492	model | metrics
Fast R-CNN R50-FPN	1x	0.140	0.035	2.6	37.8		137635226	model | metrics

COCO Instance Segmentation Baselines with Mask R-CNN

Name	lr sched	train time (s/iter)	inference time (s/im)	train mem (GB)	box AP	mask AP	model id	download

slide by Yuxin Wu

Quick Start with Model Zoo: Inference

- Pick a model: Config + Checkpoint
- Run it:

```
cfg = get_cfg()
cfg.merge_from_file("./configs/…/mask_rcnn_R_50_FPN_3x.yaml")
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.5 # set threshold for this model
# Find a model from detectron2's model zoo. You can either use the
# https://dl.fbaipublicfiles.... url, or use the following shorthand
cfg.MODEL.WEIGHTS = "detectron2://…/mask_rcnn_R_50_FPN_3x/….pkl"
predictor = DefaultPredictor(cfg)
predictions = predictor(img)
```

slide by Yuxin Wu

Quick Start with Model Zoo: Visualization

- Show outputs on images

```
from detectron2.utils import Visualizer
vis = Visualizer(img, coco_metadata)
output = vis.draw_instance_predictions(predictions["instances"])
cv2.imshow("", output.get_image())
```

slide by Yuxin Wu

Quick Start with Model Zoo: Visualization

slide by Yuxin Wu

Quick Start with Model Zoo: Inference

- Show outputs on images

```
from detectron2.utils import Visualizer
vis = Visualizer(img, coco_metadata)
output = vis.draw_instance_predictions(predictions["instances"])
cv2.imshow("", output.get_image())
```

- .. And videos (with tracking)

```
from detectron2.utils import VideoVisualizer
vis = VideoVisualizer(coco_metadata)
for frame, predictions in stream_of_predictions():
    output = vis.draw_instance_predictions(frame, predictions["instances"])
    cv2.imshow("", output.get_image())
```

Detectron2 for Advanced Users

- Write new models / Improve existing models
- New data processing
- Define new tasks/metrics

- Maintainability
- Deployment

Add Something New

- Hack inside the code: `vim detectron2`
  - Quick & flexible prototyping
  - Not scalable / maintainable

- Extend existing code: `import detectron2`
  - Some* code duplication
  - Take some time
  - Maintainable

Good research codebase should be
<u>Hackable and Extensible</u>

slide by Yuxin Wu

Hackable

- Simple abstraction
- Straightforward implementation
- Well-documented
- …even can enable global cfg

```
detectron2/
├── checkpoint
├── config
├── data
│   ├── datasets
│   ├── samplers
│   └── transforms
├── engine
├── evaluation
├── layers
├── modeling
│   ├── backbone
│   ├── meta_arch
│   ├── proposal_generator
│   └── roi_heads
├── solver
├── structures
└── utils
```

Extensible / Customizable

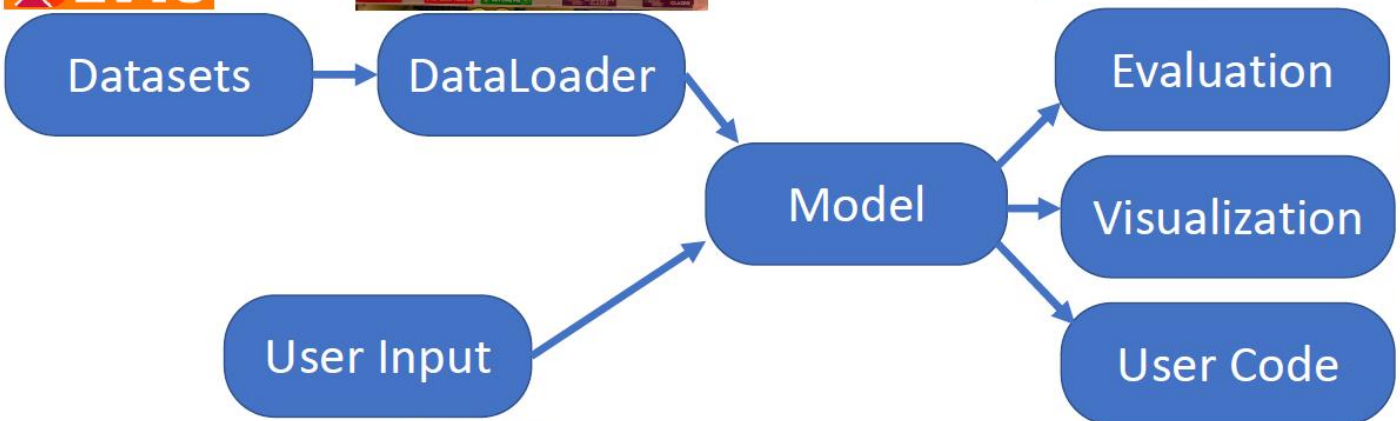
- Allow users to plug in custom
    - Models
    - Datasets
    - Data loading routines
    - Augmentations
    - Tasks/Metrics
    - Training logic
    - …

Structure of Modules

COCO Common Objects in Context

LVIS

	AP	AP50	AP75	APS	APM	APL
MMDet	0.486	0.730	0.530	0.339	0.520	0.602
Megvii (Face++)	0.485	0.737	0.532	0.298	0.507	0.641
FirstShot	0.463	0.681	0.508	0.258	0.483	0.636



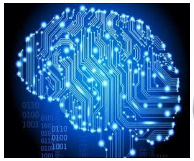
slide by Yuxin Wu

Training

- Either write your own training loop

- OR:
- Use a built-in trainer + hooks:
    - Minimal simple training needs: `SimpleTrainer`
        - Model, Dataloader, Optimizer
    - Standard training also needs:
        - Evaluation / Checkpoint / LRSchedule / … `DefaultTrainer`
        - Many users just want this standard workflow

`tools/plain_train_net.py vs. tools/train_net.py`



Detectron2 for PA1

Detectron2

https://github.com/facebookresearch/detectron2

Training & Evaluation in Command Line

We provide a script `train_net.py`, which is made to train configs.

- To train a model with `train_net.py`,

```
python train_net.py --num-gpus 4 \
    --config-file configs/R_50_1x.yaml
```

- you may need to change some parameters, e.g.:

```
python train_net.py --config-file configs/R_50_1x.yaml \
    --num-gpus 1 SOLVER.IMS_PER_BATCH 4 SOLVER.BASE_LR 0.025
```

- To evaluate a model's performance, use

```
python train_net.py --config-file configs/R_50_1x.yaml \
    --eval-only MODEL.WEIGHTS /path/to/checkpoint_file
```

For more options, see `./train_net.py -h`.

D2 Hackable: Config

detecectron2.config

Detectron2's config system uses `yaml` and `yacs`. In addition to the basic operations that access and update a config, we provide the following extra functionalities:

1. The config can have _BASE_: base.yaml field, which will **load a base config first**. Values in the base config will be overwritten in sub-configs, if there are any conflicts.
2. We provide config versioning, for backward compatibility. If your config file is versioned with a config line like VERSION: 2, detectron2 will still recognize it even if we rename some keys in the future.

Use Configs: Some basic usage of `CfgNode` object is shown below:

```
from detectron2.config import get_cfg
cfg = get_cfg()    # obtain detectron2's default config
cfg.xxx = yyy      # add new configs for your own custom components
cfg.merge_from_file("my_cfg.yaml")     # load values from a file
cfg.merge_from_list(["MODEL.WEIGHTS", "weights.pth"]) # can also load values from a list of str
print(cfg.dump()) # print formatted configs
```

D2 Hackable: Config

detecectron2.config

- The default configurations for D2 that loaded at the very first:
  - `detectron2/config/defaults.py`
- For PA1, the default configurations for FCOS:
  - `fcos/config/defaults.py`
- You can control such hyperparameters on:
  - `fcos/config/defaults.py`, or
  - `configs/*.yaml`

D2 Hackable: Datasets

detecectron2.data

- Datasets are assumed to exist in a directory `./datasets' relative to your current working directory.
- Under this directory, detectron2 expects to find datasets in the following structure:

```
coco/
    annotations/
        instances_{train,val}2017.json
        person_keypoints_{train,val}2017.json
    {train,val}2017/
        # image files that are mentioned in the corresponding json
```

D2 Hackable: Datasets

detecectron2.data

Standard Dataset Dicts

- For standard tasks (detection, instance segmentation), we load the original dataset into *list[dict]* with a specification similar to COCO's json annotations. This is our standard representation for a dataset.
- Each dict contains information about one image. The dict may have the following fields.

```
file_name: the full path to the image file.
height, width: integer. The shape of image.
image_id (str or int): a unique id that identifies this image. Used during evaluation to identify
  the images
annotations (list[dict]): each dict corresponds to annotations of one instance in this image.
  Each dict may contain the following keys:
        bbox (list[float]): list of 4 numbers representing the bounding box of the instance.
        bbox_mode (int): the format of bbox. It must be a member of structures.BoxMode.
          Currently supports: BoxMode.XYXY_ABS, BoxMode.XYWH_ABS.
        category_id (int): an integer in the range [0, num_categories) representing the category
          label. The value num_categories is reserved to represent the "background" category.
```

D2 Hackable: Models

detecectron2.modeling

- Models (and their sub-models) in detectron2 are built by functions such as `build_model`, `build_backbone`, `build_roi_heads.`
- Note that `build_model` only builds the model structure, and fill it with random parameters. To load an existing checkpoint to the model, use `DetectionCheckpointer(model).load(file_path)`.
- Detectron2 recognizes models in pytorch's `.pth` format, as well as the `.pkl` files in model zoo.
- When loading pre-trained FCOS checkpoint file, use `fcos.checkpoint.AdetCheckpointer`.
- You can use a model by just `outputs = model(inputs)`.

Model Input Format

- All builtin models take a list[dict] as the inputs. Each dict corresponds to information about one image.
- The dict may contain the following keys:

- “image”: Tensor in (C, H, W) format. The meaning of channels are defined by cfg.INPUT.FORMAT. Image normalization, if any, will be performed inside the model.

- “instances”: an Instances object, with the following fields:
    - “gt_boxes”: a Boxes object storing N boxes, one for each instance.
    - “gt_classes”: Tensor of long type, a vector of N labels, in range [0, num_categories).
    - “gt_masks”: a PolygonMasks or BitMasks object storing N masks, one for each instance.

- “height”, “width”: the **desired** output height and width, which is not necessarily the same as the height or width of the image input field. For example, the image input field might be a resized image, but you may want the outputs to be in **original** resolution.

Model Output Format

- When in **training mode**, the builtin models output a dict[str->ScalarTensor] with including all the losses.

- When in **inference mode**, the builtin models output a list[dict], one dict for each image. Each dict may contain the following fields:

- `“instances”: Instances object with the following fields:`
    - `“pred_boxes”: Boxes object storing N boxes, one for each detected instance.`
    - `“scores”: Tensor, a vector of N scores.`
    - `“pred_classes”: Tensor, a vector of N labels in range [0, num_categories).`
    - `“pred_masks”: a Tensor of shape (N, H, W), masks for each detected instance.`

D2 Hackable: Structures → Instances

detecectron2.structures.Instances

- Represents a list of instances in an image. It stores the attributes of instances (e.g., boxes, masks, labels, scores) as "fields".

- Set/Get a field:

```
instances.gt_boxes = Boxes(...)
print(instances.pred_masks) # a tensor of shape (N, H, W)
print('gt_masks' in instances)
```

- **len(instances)**: returns the number of instances.

- Indexing: **instances[indices]** will apply the indexing on all the fields and returns a new **Instances**. Typically, **indices** is a integer vector of indices, or a binary mask of length **num_instances**,

https://detectron2.readthedocs.io/modules/structures.html#detectron2.structures.Instances

D2 Hackable: Structures → Boxes

detecectron2.structures.Boxes

- Stores a list of boxes as a Nx4 `torch.Tensor`. It supports some common methods about boxes *(area, clip, nonempty, etc)*, and also behaves like a Tensor (support indexing, *to(device)*, *.device*, and iteration over all boxes)

- Set a field: tensor (Tensor[float]) – a Nx4 matrix. Each row is (x1, y1, x2, y2).

```
__init__(tensor: torch.Tensor)
```

https://detectron2.readthedocs.io/modules/structures.html#detectron2.structures.Boxes

D2 Hackable: Structures → ImageList

`detecectron2.structures.ImageList`

- Holds a list of images (of possibly varying sizes) as a single tensor. This works by padding the images to the same size, and storing in a field the original sizes of each image.

`image_sizes: each tuple is (h, w)`

- Type: list[tuple[int, int]]

`__init__(tensor: torch.Tensor, image_sizes: List[Tuple[int, int]])`

- **Parameters**:

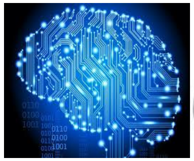
  tensor (Tensor) – of shape (N, H, W) or (N, C_1, …, C_K, H, W) where K >= 1

  image_sizes (list[tuple[int, int]]) – Each tuple is (h, w). It can be smaller than (H, W) due to padding.

`__getitem__(idx: Union[int, slice]) → torch.Tensor`

- Access the individual image in its original size.
- Returns: *Tensor*- an image of shape (H, W) or (C_1, …, C_K, H, W) where K >= 1

https://detectron2.readthedocs.io/modules/structures.html#detectron2.structures.ImageList



FCOS Implementation

Programming Assignment 1

- We provide detectron2-based skeleton code for FCOS detector.
- The followings are prepared in the skeleton code:
    - Configurations

      skeleton/configs/.yaml*

      skeleton/fcos/config/defaults.py
    - Meta-architecture containing backbone structure for FCOS

      skeleton/fcos/modeling/{backbone, meta_arch}/
    - Any other utilities related to operating FCOS detector

      Skeleton/train_net.py

      Skeleton/fcos/{checkpoint, evaluation, layers, utils}/
    - **Skeleton code** for FCOS pipelines, including FCOS heads

      skeleton/fcos/modeling/fcos/{fcos, fcos_head, fcos_losses, fcos_targets}.py
- All you have to do is to complete each part of skeleton code step-by-step.

Programming Assignment 1

- Followings are **requirements** for you to earn all credits (100) for PA1.
  - Implementation of baseline FCOS detector.
  - Train the baseline model on COCO dataset. (options for **`configs/R_50_1x.yaml`**)
  - Evaluate on COCO validation set, and record Average Precisions on your report.
  - Report qualitative detection results on some selected COCO validation images, or on any other images, using `` :class:`detectron2.utils.Visualizer` ``.

- **Do not change any other options in config files.**
  - Especially, we fix backbones and input resolutions for reproduction purpose.
  - You may change configs for \`SOLVER\` when following linear scaling rule [1].

[1] P. Goyal et al., Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour, arXiv’17

Training FCOS

- Default hyperparameters related to training schedules are set assuming that we train the model with 4-GPUs, with total batch size 16. (4 images per GPU).

- If you need to change the total batch size by some reasons (i.e., #GPUs, #imgs/GPU), you should modify the initial learning rate value and total training iterations (including what iterations to down-scale the learning rate), following the **linear scaling rule** specified in [1].

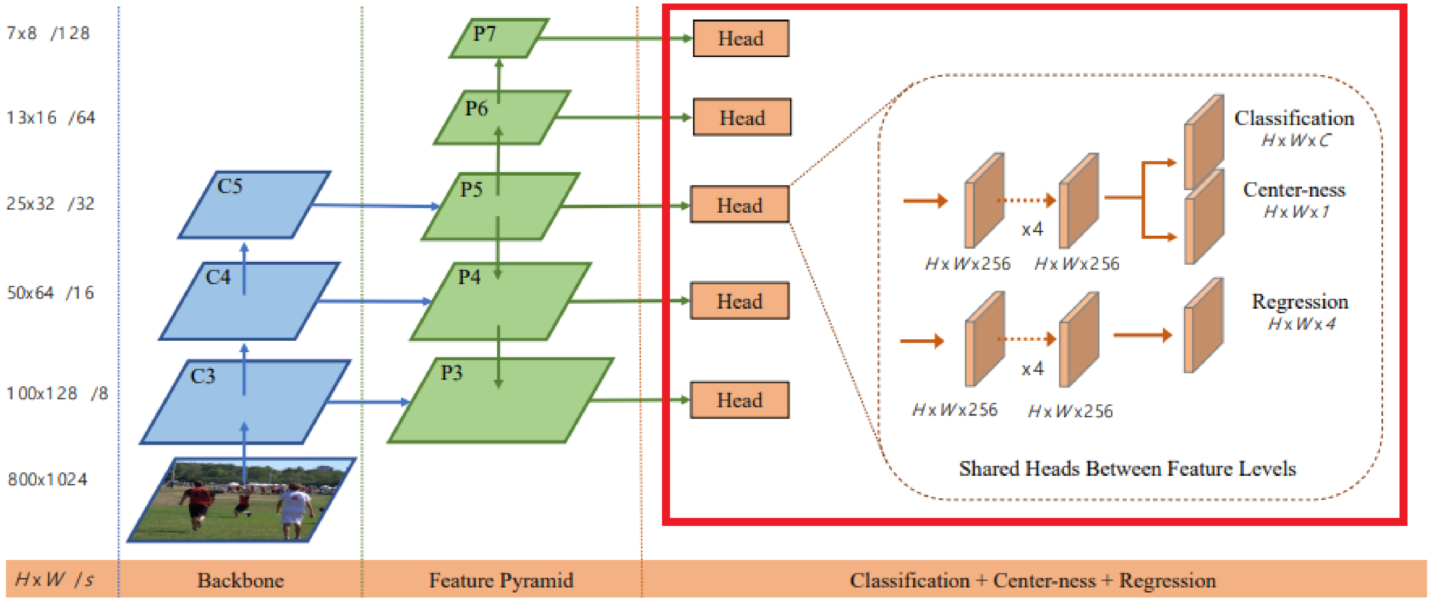
- \`SOLVER\` hyperparameters are specified in `configs/Base-FCOS.yaml`.

[1] P. Goyal et al., Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour, arXiv'17

Step 1. FCOS Head

`fcos.modeling.fcos.FCOSHead`

- FCOS head structure is as following.
  - The heads across all different feature levels share the same parameters.
- Please refer to **Figure 2** and **"Network Outputs"** on Section 3 from the FCOS paper.
- To make clear, for the final results from each of classification, centerness, and regression layer, applying activation function (i.e., sigmoid) is not needed. Proper activation function is applied when computing loss function and decoding bounding box proposals.



Step 1. FCOS Head

`fcos.modeling.fcos.FCOSHead`

- Tips
  - `` :meth:`FCOSHead.__init__` `` is implemented.
  - You need to implement the remaining methods:
    `` :meth:`FCOSHead.{_init_layers, _init_weights, forward}`. ``
  - When you define a conv2d layer, you may use `` `detectron2.layers.Conv2d` `` wrapper.
  - For weight initialization in `` :meth:`FCOSHead._init_weights`, ``
    you may use `` :func:`fcos.layers.normal_init`. ``
  - Make sure that you control each building components by the hyperparameters specified in config file automatically.

Step 2. FCOS Targets

`fcos.modeling.fcos.FCOSTargets`

- Here, you need to compute ground-truth targets needed for training FCOS:
  - Finally returns `labels` and `bbox_targets`.

- The codeflow is constructed as following.
  - `:func:`get_points`:` compute (x, y) coordinates for feature points across all feature levels back-projected into image coordinates.
  - `:func:`fcos_target`:` compute `*class labels*` and `*regression targets*` for every feature points computed in above, across all feature levels.
  - `:func:`compute_centerness_targets`:` compute centerness targets for every feature points, given bbox targets. This function is called from `:func:`FCOSLosses``.

Step 2-1. FCOS Targets: get_points

`FCOSTargets.get_points`

- `:func:`get_points`` evaluates :func:`get_points_single`` and accumulates the results per feature level.
- Complete the code of `:func:`get_points_single``.
- Hint: using `:func:`torch.meshgrid`` will be helpful.

- From FCOS paper **Section 3.1**,

For each location (x, y) on feature map $\mathrm{F_i}$,

we can map back onto the input image as $\left(\left\lfloor\frac{s}{2}\right\rfloor + xs, \left\lfloor\frac{s}{2}\right\rfloor + ys\right)$

Step 2-2. FCOS Targets: fcos_target

`FCOSTargets.fcos_target`

- `` :func:`fcos_target` `` evaluates `` :func:`fcos_target_single_image` `` and accumulates the results per image.
- Implement `` :func:`fcos_target_single_image` ``.
- Since the process of obtaining classification targets and regression targets is not intuitive, we provide line-by-line skeleton code examples.
- Replace the `*NotImplemented*` with your answers.
- Or, you may re-write the whole codes from scratch on yourself.
- We recommend you to read **Section 3.2** on FCOS paper.

Step 2-3. FCOS Targets: compute_centerness_targets

```
FCOSTargets.compute_centerness_targets
```

- Given regression targets, a tensor shape of (N, 4), compute the centerness targets.
- See **Equation (3)** and **Figure 3** from FCOS paper.

Step 3. FCOS Losses

`fcos.modeling.fcos.FCOSLosses`

- FCOS adopts two types of loss functions: `` `focal loss` `` and `` `IOU loss` `` for classification and regression task, respectively.
- For detail, see **Section 3.1 "Loss Function"** on FCOS paper.
- We provide loss function implementations, so you are free from implementing loss functions.

Step 4. Inference

`FCOS.predict_proposals`

- In inference phase, `:meth:`FCOS.predict_proposals`` predicts a complete form of detected bounding boxes with corresponding class indices from the outputs of FCOS head.

- This process includes:
  - Ranking proposals (top-k) and sorting based on class scores.
  - Score thresholding to filter out low-quality predictions.
  - Decoding regressions (l, t, r, b) into bounding boxes with corresponding classes.
  - Non-maximum suppression for removing duplicated boxes.

- See **Section 3.3** on FCOS paper describing the inference phase.

Step 4. Inference

`FCOS.predict_proposals`

- Here are some tips for implementing inference phase for FCOS detector.
  - `:meth:`FCOS.predict_proposals``
    evaluates `:meth:`FCOS.predict_proposals_single_image`` and accumulates the results per image. You need to complete this per-image operation.
  - When ranking the proposals, class scores should be multiplied by centerness scores.
  - One trick for ranking proposals is that for a single row of `` `scores` `` variable from the code, it can be regarded as total `K` number of independent predictions for all `K` classes with their scores, rather than only `one prediction` per row with the maximum score as the most likely class as the same with standard R-CNN family detectors, since the activation of classification logits is sigmoid function.
  - Inside the per feature-level loop, the predicted bounding boxes should be decoded in the form of (x1, y1, x2, y2), a tensor shape of (N, 4).
  - For the very final classification scores, applying **square root** is needed to prevent down-weighted scores (caused by multiplication with centerness scores).
  - You should utilize `:class:`Instances`` and `:class:`Boxes`` to properly construct per feature-level model outputs with the right format.
  - This processed `:class:`Instances`` object that contains detection results are accumulated across all feature levels and then forwarded to nms-layer which is given.

Extra Credits

- You can earn extra credits if you implement methods corresponding to **"Improvements" on Table 3** on FCOS paper, including 'w/ GN' option and make comparisons quantitatively/qualitatively with the baseline model.
(For GN, use `:class:`torch.nn.GroupNorm``.)

- When training the improved version, use `configs/R_50_1x_improve.yaml`.
  - You may freely modify the options for the improvements as far as you implemented .

- You may compare the only your final version of the improved model with the baseline if your GPU environment is limited.

ExtraExtra Credits

- Expand the detection task to Instance Segmentation task.
- FCOS detector as a proposal generator for `GeneralizedRCNN` framework.
- Attach a ROI pooling layer (i.e., ROIAlign) and a standard mask head implemented in Detectron2 in a cascading manner after FCOS bbox detections.
  - Use FCOS detection results for sampling ROIs that will be provided to `mask roi head`.
  - Also, regard those detection results as the final bbox results. (No additional bbox head)
- In this part, you should generate proper configuration file for training. Also making some modifications on original D2 codes will be needed.
  - For modification of codes, you write a new custom model code that overrides the existing relevant components (i.e., ROIHeads, MaskHead, Sampler, …) from D2, and make changes on your code wherein only required.
- When FCOS-based instance segmentation task is implemented,
  - Report Mask Average Precisions
  - Compare with standard `R-50-FPN Mask R-CNN` model quantitatively/qualitatively.
  - You may download the `COCO-pretrained Mask R-CNN weights` from D2 model zoo and make comparisons with your implemented Instance Segmentation model.

Cautions and Comment

- **Do not modify any files or folders** except for python files located in the path: `skeleton/fcos/modeling/fcos/*.py`
- Exceptionally, you may define helper functions on existing files or new `.py` file for your convenience.
- There will be partial points on every steps, so try to do your best.
- Collaboration with other students is prohibited. Please do it yourself and comment on your code in order to show your understanding.

PA1 Submission

- **Due**: **May 12th 11:59PM** (No deadline extension)
- **To**: Youngtaek Oh (youngtaek.oh [at] kaist.ac.kr)
- **Submission should include**:
    - Report (`.docx` or `.pdf`)
    - Source code (do not include dataset folder)
    - Only one `.pth`checkpoint that yields the best AP.
      You should mention which method your result corresponds to in Table 3.
    - Zip your all includings named as `PA1_{StudentID}_{NAME}.zip`,
      i.e., `PA1_20193358_YoungtaekOh.zip`.

- **The report should include**:
    - Your understanding of each step of FCOS implementation.
    - Reproduced AP for the baseline model (on Table 3, 4th row of FCOS paper)
    - Visualization of detection results from the baseline model.
    - (Extra) Average Precisions for your improved model (Table 3 Improvements) and comparisons with the baseline model.
    - (ExtraExtra) The mask Average Precisions of your Instance Segmentation model and comparisons with `R-50-FPN Mask R-CNN` model.

Revision History

Revision	Date	Description
0	2020-04-14	Release
1	2020-04-26	p. 39, clarification of FCOS head predictions.
2	2020-05-01	1. Fixing a bug on :func: `fcos_target_single_image` in `fcos_targets.py` 2. Fixing typo on docstring.