detectron2-onnx

April 9, 2021

1 Detectron2 onnx export and inference

To gain some speed, especially in embedded devices such as the raspberry pi. Deploying a detectron2 model and run the inference on a CPU based backend like the caffe2 onnx backend is a must.

This notebooks describes how to do this. It is based on the official colab.

Run this notebook inside the detectron2 repository

2 Install detectron2

[1]:
```
import torch, torchvision
print(torch.__version__, torch.cuda.is_available())
```

```
1.8.0 False
```

[2]:
```
# Some basic setup:
# Setup detectron2 logger
import detectron2
from detectron2.utils.logger import setup_logger
setup_logger()

import time

# import some common libraries
import numpy as np
import os, json, cv2, random
import matplotlib.pyplot as plt

plt.rcParams['figure.dpi'] = 200

def cv2_imshow(im):
    plt.imshow(cv2.cvtColor(im, cv2.COLOR_BGR2RGB))

# import some common detectron2 utilities
from detectron2 import model_zoo
from detectron2.engine import DefaultPredictor
from detectron2.config import get_cfg
```

```
from detectron2.utils.visualizer import Visualizer
from detectron2.data import MetadataCatalog, DatasetCatalog

from detectron2.data import build_detection_test_loader

import detectron2.data.transforms as T
```

3 Run a pre-trained detectron2 model

We first download an image from the COCO dataset:

[ ]:
```
# prepare the coco test dataset
!wget http://images.cocodataset.org/zips/val2017.zip
!mkdir -p datasets/coco/
!unzip val2017.zip -d datasets/coco/
./datasets/prepare_for_tests.sh
!mv datasets/coco/annotations/instances_val2017_100.json datasets/coco/
 ↪annotations/instances_val2017.json
```

Create a detectron2 config. Choose any model from the model zoo.

[3]:
```
model_config = "COCO-Detection/faster_rcnn_R_50_FPN_3x.yaml"
```

[4]:
```
cfg = get_cfg()

# add project-specific config (e.g., TensorMask) here if you're not running a␣
 ↪model in detectron2's core library
cfg.merge_from_file(model_zoo.get_config_file(model_config))
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.5  # set threshold for this model

# Find a model from detectron2's model zoo. You can use the https://dl.
 ↪fbaipublicfiles... url as well
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url(model_config)
cfg.MODEL.DEVICE = 'cpu'
```

[5]:
```
# Load an image from the coco dataset.
original_image = cv2.imread('datasets/coco/val2017/000000023937.jpg')
```

[6]:
```
# Use the detectron2 `DefaultPredictor` to run inference on this image.
predictor = DefaultPredictor(cfg)
outputs = predictor(original_image)
```

[7]:
```
# We can use `Visualizer` to draw the predictions on the image.
v = Visualizer(original_image[:, :, ::-1], MetadataCatalog.get(cfg.DATASETS.
 ↪TRAIN[0]), scale=1.2)
out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
```

```
cv2_imshow(out.get_image()[:, :, ::-1])
```

4 Export the model as onnx

Detectron2 already provides a script to export onnx models, use it with:

```
./tools/deploy/export_model.py --config-file configs/COCO-InstanceSegmentation/mask_rcnn_R_50_
    --output ./output \
    --export-method caffe2_tracing \
    --format onnx \
    MODEL.WEIGHTS detectron2://COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x/137849600/model_
    MODEL.DEVICE cpu
```

The following steps show how the onnx tracing is done.

[8]:
```
from detectron2.checkpoint import DetectionCheckpointer
from detectron2.modeling import build_model

from detectron2.export import Caffe2Tracer

from torchvision.transforms import transforms

import onnx
```

[9]:
```
# build the model with the config used before
torch_model = build_model(cfg)
```

```
DetectionCheckpointer(torch_model).resume_or_load(cfg.MODEL.WEIGHTS)
torch_model.eval()
```

[9]:
```
GeneralizedRCNN(
  (backbone): FPN(
    (fpn_lateral2): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (fpn_lateral3): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (fpn_lateral4): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output4): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (fpn_lateral5): Conv2d(2048, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output5): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (top_block): LastLevelMaxPool()
    (bottom_up): ResNet(
      (stem): BasicStem(
        (conv1): Conv2d(
          3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False
          (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
        )
      )
      (res2): Sequential(
        (0): BottleneckBlock(
          (shortcut): Conv2d(
            64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv1): Conv2d(
            64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
          )
          (conv2): Conv2d(
            64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
          )
          (conv3): Conv2d(
            64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
        )
        (1): BottleneckBlock(
          (conv1): Conv2d(
```

```
            256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
          )
          (conv2): Conv2d(
            64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
          )
          (conv3): Conv2d(
            64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
        )
        (2): BottleneckBlock(
          (conv1): Conv2d(
            256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
          )
          (conv2): Conv2d(
            64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
          )
          (conv3): Conv2d(
            64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
        )
      )
      (res3): Sequential(
        (0): BottleneckBlock(
          (shortcut): Conv2d(
            256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
          )
          (conv1): Conv2d(
            256, 128, kernel_size=(1, 1), stride=(2, 2), bias=False
            (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
          )
          (conv2): Conv2d(
            128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
          )
          (conv3): Conv2d(
            128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
```

```
          )
        )
        (1): BottleneckBlock(
          (conv1): Conv2d(
            512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
          )
          (conv2): Conv2d(
            128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
          )
          (conv3): Conv2d(
            128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
          )
        )
        (2): BottleneckBlock(
          (conv1): Conv2d(
            512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
          )
          (conv2): Conv2d(
            128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
          )
          (conv3): Conv2d(
            128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
          )
        )
        (3): BottleneckBlock(
          (conv1): Conv2d(
            512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
          )
          (conv2): Conv2d(
            128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
          )
          (conv3): Conv2d(
            128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
          )
        )
```

```
      )
      (res4): Sequential(
        (0): BottleneckBlock(
          (shortcut): Conv2d(
            512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=False
            (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
          )
          (conv1): Conv2d(
            512, 256, kernel_size=(1, 1), stride=(2, 2), bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv2): Conv2d(
            256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv3): Conv2d(
            256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
          )
        )
        (1): BottleneckBlock(
          (conv1): Conv2d(
            1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv2): Conv2d(
            256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv3): Conv2d(
            256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
          )
        )
        (2): BottleneckBlock(
          (conv1): Conv2d(
            1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv2): Conv2d(
            256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv3): Conv2d(
```

```
            256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
          )
        )
        (3): BottleneckBlock(
          (conv1): Conv2d(
            1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv2): Conv2d(
            256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv3): Conv2d(
            256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
          )
        )
        (4): BottleneckBlock(
          (conv1): Conv2d(
            1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv2): Conv2d(
            256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv3): Conv2d(
            256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
          )
        )
        (5): BottleneckBlock(
          (conv1): Conv2d(
            1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv2): Conv2d(
            256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv3): Conv2d(
            256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
```

```
          )
        )
      )
      (res5): Sequential(
        (0): BottleneckBlock(
          (shortcut): Conv2d(
            1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False
            (norm): FrozenBatchNorm2d(num_features=2048, eps=1e-05)
          )
          (conv1): Conv2d(
            1024, 512, kernel_size=(1, 1), stride=(2, 2), bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
          )
          (conv2): Conv2d(
            512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
          )
          (conv3): Conv2d(
            512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=2048, eps=1e-05)
          )
        )
        (1): BottleneckBlock(
          (conv1): Conv2d(
            2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
          )
          (conv2): Conv2d(
            512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
          )
          (conv3): Conv2d(
            512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=2048, eps=1e-05)
          )
        )
        (2): BottleneckBlock(
          (conv1): Conv2d(
            2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
          )
          (conv2): Conv2d(
            512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
```

```
          )
          (conv3): Conv2d(
            512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=2048, eps=1e-05)
          )
        )
      )
    )
  )
  (proposal_generator): RPN(
    (rpn_head): StandardRPNHead(
      (conv): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
      (objectness_logits): Conv2d(256, 3, kernel_size=(1, 1), stride=(1, 1))
      (anchor_deltas): Conv2d(256, 12, kernel_size=(1, 1), stride=(1, 1))
    )
    (anchor_generator): DefaultAnchorGenerator(
      (cell_anchors): BufferList()
    )
  )
  (roi_heads): StandardROIHeads(
    (box_pooler): ROIPooler(
      (level_poolers): ModuleList(
        (0): ROIAlign(output_size=(7, 7), spatial_scale=0.25, sampling_ratio=0,
aligned=True)
        (1): ROIAlign(output_size=(7, 7), spatial_scale=0.125, sampling_ratio=0,
aligned=True)
        (2): ROIAlign(output_size=(7, 7), spatial_scale=0.0625,
sampling_ratio=0, aligned=True)
        (3): ROIAlign(output_size=(7, 7), spatial_scale=0.03125,
sampling_ratio=0, aligned=True)
      )
    )
    (box_head): FastRCNNConvFCHead(
      (flatten): Flatten(start_dim=1, end_dim=-1)
      (fc1): Linear(in_features=12544, out_features=1024, bias=True)
      (fc_relu1): ReLU()
      (fc2): Linear(in_features=1024, out_features=1024, bias=True)
      (fc_relu2): ReLU()
    )
    (box_predictor): FastRCNNOutputLayers(
      (cls_score): Linear(in_features=1024, out_features=81, bias=True)
      (bbox_pred): Linear(in_features=1024, out_features=320, bias=True)
    )
  )
)
```

[10]:
```
# https://github.com/facebookresearch/detectron2/blob/master/tools/deploy/
 ↪export_model.py#L45
from detectron2.data import build_detection_test_loader

# get a sample data
data_loader = build_detection_test_loader(cfg, cfg.DATASETS.TEST[0])
first_batch = next(iter(data_loader))

# https://github.com/facebookresearch/detectron2/blob/master/tools/deploy/
 ↪export_model.py#L45
tracer = Caffe2Tracer(
    cfg,
    torch_model,
    first_batch
)

onnx_model = tracer.export_onnx()
onnx.save(onnx_model, "model.onnx")
```

```
[03/18 19:26:47 d2.data.datasets.coco]: Loaded 100 images in COCO
format from datasets/coco/annotations/instances_val2017.json
[03/18 19:26:47 d2.data.build]: Distribution of instances among all 80
categories:
```

category	#instances	category	#instances	category	#instances
person	341	bicycle	10	car	51
motorcycle	23	airplane	0	bus	10
train	2	truck	4	boat	13
traffic light	9	fire hydrant	5	stop sign	1
parking meter	0	bench	14	bird	13
cat	2	dog	4	horse	5
sheep	0	cow	1	elephant	3
bear	6	zebra	16	giraffe	2
backpack	8	umbrella	8	handbag	12
tie	11	suitcase	0	frisbee	9
skis	8	snowboard	1	sports ball	4
kite	19	baseball bat	2	baseball gl..	1
skateboard	1	surfboard	3	tennis racket	9
bottle	16	wine glass	5	cup	15
fork	2	knife	0	spoon	1
bowl	7	banana	1	apple	2

```
[03/18 19:26:47 d2.data.dataset_mapper]: [DatasetMapper] Augmentations
used in inference: [ResizeShortestEdge(short_edge_length=(800, 800),
max_size=1333, sample_style='choice')]
[03/18 19:26:47 d2.data.common]: Serializing 100 elements to byte
tensors and concatenating them all …
[03/18 19:26:47 d2.data.common]: Serialized dataset takes 0.46 MiB

/home/users/fassbinb/.local/lib/python3.8/site-packages/torch/onnx/utils.py:262:
UserWarning: `add_node_names' can be set to True only when
'operator_export_type' is `ONNX`. Since 'operator_export_type' is not set to
'ONNX', `add_node_names` argument will be ignored.
  warnings.warn("`{}' can be set to True only when 'operator_export_type' is "
/home/users/fassbinb/.local/lib/python3.8/site-packages/torch/onnx/utils.py:262:
UserWarning: `do_constant_folding' can be set to True only when
'operator_export_type' is `ONNX`. Since 'operator_export_type' is not set to
'ONNX', `do_constant_folding` argument will be ignored.
  warnings.warn("`{}' can be set to True only when 'operator_export_type' is "
/home/users/fassbinb/Work/detectron2/detectron2/export/c10.py:31: TracerWarning:
Converting a tensor to a Python boolean might cause the trace to be incorrect.
We can't record the data flow of Python values, so this value will be treated as
a constant in the future. This means that the trace might not generalize to
other inputs!
  assert tensor.dim() == 2 and tensor.size(-1) in [4, 5, 6], tensor.size()
/home/users/fassbinb/Work/detectron2/detectron2/export/c10.py:385:
TracerWarning: Converting a tensor to a Python boolean might cause the trace to
be incorrect. We can't record the data flow of Python values, so this value will
be treated as a constant in the future. This means that the trace might not
generalize to other inputs!
  if num_classes + 1 == class_logits.shape[1]:
/home/users/fassbinb/Work/detectron2/detectron2/export/c10.py:394:
TracerWarning: Converting a tensor to a Python boolean might cause the trace to
be incorrect. We can't record the data flow of Python values, so this value will
be treated as a constant in the future. This means that the trace might not
generalize to other inputs!
  assert box_regression.shape[1] % box_dim == 0
/home/users/fassbinb/Work/detectron2/detectron2/export/c10.py:401:
TracerWarning: Converting a tensor to a Python boolean might cause the trace to
be incorrect. We can't record the data flow of Python values, so this value will
be treated as a constant in the future. This means that the trace might not
generalize to other inputs!
  if input_tensor_mode:
/home/users/fassbinb/Work/detectron2/detectron2/export/c10.py:433:
TracerWarning: Converting a tensor to a Python boolean might cause the trace to
be incorrect. We can't record the data flow of Python values, so this value will
be treated as a constant in the future. This means that the trace might not
generalize to other inputs!
  nms_outputs = torch.ops._caffe2.BoxWithNMSLimit(
/home/users/fassbinb/.local/lib/python3.8/site-packages/torch/tensor.py:587:
```

```
RuntimeWarning: Iterating over a tensor might cause the trace to be incorrect.
Passing a tensor of different shape won't change the number of iterations
executed (and might lead to errors or silently give incorrect results).
  warnings.warn('Iterating over a tensor might cause the trace to be incorrect.
'
/home/users/fassbinb/Work/detectron2/detectron2/export/c10.py:462:
TracerWarning: Converting a tensor to a Python number might cause the trace to
be incorrect. We can't record the data flow of Python values, so this value will
be treated as a constant in the future. This means that the trace might not
generalize to other inputs!
  for i, b in enumerate(int(x.item()) for x in roi_batch_splits_nms)
```

[11]:
```
# the model can be loaded using the Caffe2 backend (optimized for cpu␣
 ↪inference) and
# do predicions without detectron2
import caffe2.python.onnx.backend as backend

model = onnx.load("model.onnx")  # or use the `onnx_model` from above
onnx.checker.check_model(model)

# To print a human readable representation of the graph use:
print(onnx.helper.printable_graph(model.graph))
```

```
graph torch-jit-export (
  %data[UINT8, 1x3x800x1216]
  %im_info[FLOAT, 1x3]
) optional inputs with matching initializers (
  %_wrapped_model.backbone.fpn_lateral2.weight[FLOAT, 256x256x1x1]
  %_wrapped_model.backbone.fpn_lateral2.bias[FLOAT, 256]
  %_wrapped_model.backbone.fpn_output2.weight[FLOAT, 256x256x3x3]
  %_wrapped_model.backbone.fpn_output2.bias[FLOAT, 256]
  %_wrapped_model.backbone.fpn_lateral3.weight[FLOAT, 256x512x1x1]
  %_wrapped_model.backbone.fpn_lateral3.bias[FLOAT, 256]
  %_wrapped_model.backbone.fpn_output3.weight[FLOAT, 256x256x3x3]
  %_wrapped_model.backbone.fpn_output3.bias[FLOAT, 256]
  %_wrapped_model.backbone.fpn_lateral4.weight[FLOAT, 256x1024x1x1]
  %_wrapped_model.backbone.fpn_lateral4.bias[FLOAT, 256]
  %_wrapped_model.backbone.fpn_output4.weight[FLOAT, 256x256x3x3]
  %_wrapped_model.backbone.fpn_output4.bias[FLOAT, 256]
  %_wrapped_model.backbone.fpn_lateral5.weight[FLOAT, 256x2048x1x1]
  %_wrapped_model.backbone.fpn_lateral5.bias[FLOAT, 256]
  %_wrapped_model.backbone.fpn_output5.weight[FLOAT, 256x256x3x3]
  %_wrapped_model.backbone.fpn_output5.bias[FLOAT, 256]
  %_wrapped_model.proposal_generator.rpn_head.conv.weight[FLOAT, 256x256x3x3]
  %_wrapped_model.proposal_generator.rpn_head.conv.bias[FLOAT, 256]
  %_wrapped_model.proposal_generator.rpn_head.objectness_logits.weight[FLOAT,
3x256x1x1]
  %_wrapped_model.proposal_generator.rpn_head.objectness_logits.bias[FLOAT, 3]
```

```
  %_wrapped_model.proposal_generator.rpn_head.anchor_deltas.weight[FLOAT,
12x256x1x1]
  %_wrapped_model.proposal_generator.rpn_head.anchor_deltas.bias[FLOAT, 12]
  %_wrapped_model.proposal_generator.anchor_generator.cell_anchors.0[FLOAT, 3x4]
  %_wrapped_model.proposal_generator.anchor_generator.cell_anchors.1[FLOAT, 3x4]
  %_wrapped_model.proposal_generator.anchor_generator.cell_anchors.2[FLOAT, 3x4]
  %_wrapped_model.proposal_generator.anchor_generator.cell_anchors.3[FLOAT, 3x4]
  %_wrapped_model.proposal_generator.anchor_generator.cell_anchors.4[FLOAT, 3x4]
  %_wrapped_model.roi_heads.box_head.fc1.weight[FLOAT, 1024x12544]
  %_wrapped_model.roi_heads.box_head.fc1.bias[FLOAT, 1024]
  %_wrapped_model.roi_heads.box_head.fc2.weight[FLOAT, 1024x1024]
  %_wrapped_model.roi_heads.box_head.fc2.bias[FLOAT, 1024]
  %_wrapped_model.roi_heads.box_predictor.cls_score.weight[FLOAT, 81x1024]
  %_wrapped_model.roi_heads.box_predictor.cls_score.bias[FLOAT, 81]
  %_wrapped_model.roi_heads.box_predictor.bbox_pred.weight[FLOAT, 320x1024]
  %_wrapped_model.roi_heads.box_predictor.bbox_pred.bias[FLOAT, 320]
  %560[FLOAT, 64x3x7x7]
  %561[FLOAT, 64]
  %563[FLOAT, 64x64x1x1]
  %564[FLOAT, 64]
  %566[FLOAT, 64x64x3x3]
  %567[FLOAT, 64]
  %569[FLOAT, 256x64x1x1]
  %570[FLOAT, 256]
  %572[FLOAT, 256x64x1x1]
  %573[FLOAT, 256]
  %575[FLOAT, 64x256x1x1]
  %576[FLOAT, 64]
  %578[FLOAT, 64x64x3x3]
  %579[FLOAT, 64]
  %581[FLOAT, 256x64x1x1]
  %582[FLOAT, 256]
  %584[FLOAT, 64x256x1x1]
  %585[FLOAT, 64]
  %587[FLOAT, 64x64x3x3]
  %588[FLOAT, 64]
  %590[FLOAT, 256x64x1x1]
  %591[FLOAT, 256]
  %593[FLOAT, 128x256x1x1]
  %594[FLOAT, 128]
  %596[FLOAT, 128x128x3x3]
  %597[FLOAT, 128]
  %599[FLOAT, 512x128x1x1]
  %600[FLOAT, 512]
  %602[FLOAT, 512x256x1x1]
  %603[FLOAT, 512]
  %605[FLOAT, 128x512x1x1]
  %606[FLOAT, 128]
```

```
%608[FLOAT, 128x128x3x3]
%609[FLOAT, 128]
%611[FLOAT, 512x128x1x1]
%612[FLOAT, 512]
%614[FLOAT, 128x512x1x1]
%615[FLOAT, 128]
%617[FLOAT, 128x128x3x3]
%618[FLOAT, 128]
%620[FLOAT, 512x128x1x1]
%621[FLOAT, 512]
%623[FLOAT, 128x512x1x1]
%624[FLOAT, 128]
%626[FLOAT, 128x128x3x3]
%627[FLOAT, 128]
%629[FLOAT, 512x128x1x1]
%630[FLOAT, 512]
%632[FLOAT, 256x512x1x1]
%633[FLOAT, 256]
%635[FLOAT, 256x256x3x3]
%636[FLOAT, 256]
%638[FLOAT, 1024x256x1x1]
%639[FLOAT, 1024]
%641[FLOAT, 1024x512x1x1]
%642[FLOAT, 1024]
%644[FLOAT, 256x1024x1x1]
%645[FLOAT, 256]
%647[FLOAT, 256x256x3x3]
%648[FLOAT, 256]
%650[FLOAT, 1024x256x1x1]
%651[FLOAT, 1024]
%653[FLOAT, 256x1024x1x1]
%654[FLOAT, 256]
%656[FLOAT, 256x256x3x3]
%657[FLOAT, 256]
%659[FLOAT, 1024x256x1x1]
%660[FLOAT, 1024]
%662[FLOAT, 256x1024x1x1]
%663[FLOAT, 256]
%665[FLOAT, 256x256x3x3]
%666[FLOAT, 256]
%668[FLOAT, 1024x256x1x1]
%669[FLOAT, 1024]
%671[FLOAT, 256x1024x1x1]
%672[FLOAT, 256]
%674[FLOAT, 256x256x3x3]
%675[FLOAT, 256]
%677[FLOAT, 1024x256x1x1]
%678[FLOAT, 1024]
```

```
  %680[FLOAT, 256x1024x1x1]
  %681[FLOAT, 256]
  %683[FLOAT, 256x256x3x3]
  %684[FLOAT, 256]
  %686[FLOAT, 1024x256x1x1]
  %687[FLOAT, 1024]
  %689[FLOAT, 512x1024x1x1]
  %690[FLOAT, 512]
  %692[FLOAT, 512x512x3x3]
  %693[FLOAT, 512]
  %695[FLOAT, 2048x512x1x1]
  %696[FLOAT, 2048]
  %698[FLOAT, 2048x1024x1x1]
  %699[FLOAT, 2048]
  %701[FLOAT, 512x2048x1x1]
  %702[FLOAT, 512]
  %704[FLOAT, 512x512x3x3]
  %705[FLOAT, 512]
  %707[FLOAT, 2048x512x1x1]
  %708[FLOAT, 2048]
  %710[FLOAT, 512x2048x1x1]
  %711[FLOAT, 512]
  %713[FLOAT, 512x512x3x3]
  %714[FLOAT, 512]
  %716[FLOAT, 2048x512x1x1]
  %717[FLOAT, 2048]
) {
  %302 = AliasWithName[is_backward = 0, name = 'data'](%data)
  %303 = AliasWithName[is_backward = 0, name = 'im_info'](%im_info)
  %304 = Cast[to = 1](%302)
  %305 = Constant[value = <Tensor>]()
  %306 = Sub(%304, %305)
  %307 = Constant[value = <Tensor>]()
  %308 = Div(%306, %307)
  %309 = AliasWithName[is_backward = 0, name = 'normalized_data'](%308)
  %559 = Conv[dilations = [1, 1], group = 1, kernel_shape = [7, 7], pads = [3,
3, 3, 3], strides = [2, 2]](%309, %560, %561)
  %312 = Relu(%559)
  %313 = MaxPool[kernel_shape = [3, 3], pads = [1, 1, 1, 1], strides = [2,
2]](%312)
  %562 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%313, %563, %564)
  %316 = Relu(%562)
  %565 = Conv[dilations = [1, 1], group = 1, kernel_shape = [3, 3], pads = [1,
1, 1, 1], strides = [1, 1]](%316, %566, %567)
  %319 = Relu(%565)
  %568 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%319, %569, %570)
```

```
  %571 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%313, %572, %573)
  %324 = Add(%568, %571)
  %325 = Relu(%324)
  %574 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%325, %575, %576)
  %328 = Relu(%574)
  %577 = Conv[dilations = [1, 1], group = 1, kernel_shape = [3, 3], pads = [1,
1, 1, 1], strides = [1, 1]](%328, %578, %579)
  %331 = Relu(%577)
  %580 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%331, %581, %582)
  %334 = Add(%580, %325)
  %335 = Relu(%334)
  %583 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%335, %584, %585)
  %338 = Relu(%583)
  %586 = Conv[dilations = [1, 1], group = 1, kernel_shape = [3, 3], pads = [1,
1, 1, 1], strides = [1, 1]](%338, %587, %588)
  %341 = Relu(%586)
  %589 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%341, %590, %591)
  %344 = Add(%589, %335)
  %345 = Relu(%344)
  %592 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [2, 2]](%345, %593, %594)
  %348 = Relu(%592)
  %595 = Conv[dilations = [1, 1], group = 1, kernel_shape = [3, 3], pads = [1,
1, 1, 1], strides = [1, 1]](%348, %596, %597)
  %351 = Relu(%595)
  %598 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%351, %599, %600)
  %601 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [2, 2]](%345, %602, %603)
  %356 = Add(%598, %601)
  %357 = Relu(%356)
  %604 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%357, %605, %606)
  %360 = Relu(%604)
  %607 = Conv[dilations = [1, 1], group = 1, kernel_shape = [3, 3], pads = [1,
1, 1, 1], strides = [1, 1]](%360, %608, %609)
  %363 = Relu(%607)
  %610 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%363, %611, %612)
  %366 = Add(%610, %357)
  %367 = Relu(%366)
  %613 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%367, %614, %615)
```

```
  %370 = Relu(%613)
  %616 = Conv[dilations = [1, 1], group = 1, kernel_shape = [3, 3], pads = [1,
1, 1, 1], strides = [1, 1]](%370, %617, %618)
  %373 = Relu(%616)
  %619 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%373, %620, %621)
  %376 = Add(%619, %367)
  %377 = Relu(%376)
  %622 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%377, %623, %624)
  %380 = Relu(%622)
  %625 = Conv[dilations = [1, 1], group = 1, kernel_shape = [3, 3], pads = [1,
1, 1, 1], strides = [1, 1]](%380, %626, %627)
  %383 = Relu(%625)
  %628 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%383, %629, %630)
  %386 = Add(%628, %377)
  %387 = Relu(%386)
  %631 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [2, 2]](%387, %632, %633)
  %390 = Relu(%631)
  %634 = Conv[dilations = [1, 1], group = 1, kernel_shape = [3, 3], pads = [1,
1, 1, 1], strides = [1, 1]](%390, %635, %636)
  %393 = Relu(%634)
  %637 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%393, %638, %639)
  %640 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [2, 2]](%387, %641, %642)
  %398 = Add(%637, %640)
  %399 = Relu(%398)
  %643 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%399, %644, %645)
  %402 = Relu(%643)
  %646 = Conv[dilations = [1, 1], group = 1, kernel_shape = [3, 3], pads = [1,
1, 1, 1], strides = [1, 1]](%402, %647, %648)
  %405 = Relu(%646)
  %649 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%405, %650, %651)
  %408 = Add(%649, %399)
  %409 = Relu(%408)
  %652 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%409, %653, %654)
  %412 = Relu(%652)
  %655 = Conv[dilations = [1, 1], group = 1, kernel_shape = [3, 3], pads = [1,
1, 1, 1], strides = [1, 1]](%412, %656, %657)
  %415 = Relu(%655)
  %658 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%415, %659, %660)
```

```
  %418 = Add(%658, %409)
  %419 = Relu(%418)
  %661 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%419, %662, %663)
  %422 = Relu(%661)
  %664 = Conv[dilations = [1, 1], group = 1, kernel_shape = [3, 3], pads = [1,
1, 1, 1], strides = [1, 1]](%422, %665, %666)
  %425 = Relu(%664)
  %667 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%425, %668, %669)
  %428 = Add(%667, %419)
  %429 = Relu(%428)
  %670 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%429, %671, %672)
  %432 = Relu(%670)
  %673 = Conv[dilations = [1, 1], group = 1, kernel_shape = [3, 3], pads = [1,
1, 1, 1], strides = [1, 1]](%432, %674, %675)
  %435 = Relu(%673)
  %676 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%435, %677, %678)
  %438 = Add(%676, %429)
  %439 = Relu(%438)
  %679 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%439, %680, %681)
  %442 = Relu(%679)
  %682 = Conv[dilations = [1, 1], group = 1, kernel_shape = [3, 3], pads = [1,
1, 1, 1], strides = [1, 1]](%442, %683, %684)
  %445 = Relu(%682)
  %685 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%445, %686, %687)
  %448 = Add(%685, %439)
  %449 = Relu(%448)
  %688 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [2, 2]](%449, %689, %690)
  %452 = Relu(%688)
  %691 = Conv[dilations = [1, 1], group = 1, kernel_shape = [3, 3], pads = [1,
1, 1, 1], strides = [1, 1]](%452, %692, %693)
  %455 = Relu(%691)
  %694 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%455, %695, %696)
  %697 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [2, 2]](%449, %698, %699)
  %460 = Add(%694, %697)
  %461 = Relu(%460)
  %700 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%461, %701, %702)
  %464 = Relu(%700)
  %703 = Conv[dilations = [1, 1], group = 1, kernel_shape = [3, 3], pads = [1,
```

```
1, 1, 1], strides = [1, 1]](%464, %704, %705)
  %467 = Relu(%703)
  %706 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%467, %707, %708)
  %470 = Add(%706, %461)
  %471 = Relu(%470)
  %709 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%471, %710, %711)
  %474 = Relu(%709)
  %712 = Conv[dilations = [1, 1], group = 1, kernel_shape = [3, 3], pads = [1,
1, 1, 1], strides = [1, 1]](%474, %713, %714)
  %477 = Relu(%712)
  %715 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%477, %716, %717)
  %480 = Add(%715, %471)
  %481 = Relu(%480)
  %482 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%481, %_wrapped_model.backbone.fpn_lateral5.weight,
%_wrapped_model.backbone.fpn_lateral5.bias)
  %483 = Conv[dilations = [1, 1], group = 1, kernel_shape = [3, 3], pads = [1,
1, 1, 1], strides = [1, 1]](%482, %_wrapped_model.backbone.fpn_output5.weight,
%_wrapped_model.backbone.fpn_output5.bias)
  %484 = ResizeNearest[height_scale = 2, order = 'NCHW', width_scale = 2](%482)
  %485 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%449, %_wrapped_model.backbone.fpn_lateral4.weight,
%_wrapped_model.backbone.fpn_lateral4.bias)
  %486 = Add(%485, %484)
  %487 = Conv[dilations = [1, 1], group = 1, kernel_shape = [3, 3], pads = [1,
1, 1, 1], strides = [1, 1]](%486, %_wrapped_model.backbone.fpn_output4.weight,
%_wrapped_model.backbone.fpn_output4.bias)
  %488 = ResizeNearest[height_scale = 2, order = 'NCHW', width_scale = 2](%486)
  %489 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%387, %_wrapped_model.backbone.fpn_lateral3.weight,
%_wrapped_model.backbone.fpn_lateral3.bias)
  %490 = Add(%489, %488)
  %491 = Conv[dilations = [1, 1], group = 1, kernel_shape = [3, 3], pads = [1,
1, 1, 1], strides = [1, 1]](%490, %_wrapped_model.backbone.fpn_output3.weight,
%_wrapped_model.backbone.fpn_output3.bias)
  %492 = ResizeNearest[height_scale = 2, order = 'NCHW', width_scale = 2](%490)
  %493 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%345, %_wrapped_model.backbone.fpn_lateral2.weight,
%_wrapped_model.backbone.fpn_lateral2.bias)
  %494 = Add(%493, %492)
  %495 = Conv[dilations = [1, 1], group = 1, kernel_shape = [3, 3], pads = [1,
1, 1, 1], strides = [1, 1]](%494, %_wrapped_model.backbone.fpn_output2.weight,
%_wrapped_model.backbone.fpn_output2.bias)
  %496 = MaxPool[kernel_shape = [1, 1], pads = [0, 0, 0, 0], strides = [2,
2]](%483)
```

```
  %497 = Conv[dilations = [1, 1], group = 1, kernel_shape = [3, 3], pads = [1,
1, 1, 1], strides = [1, 1]](%495,
%_wrapped_model.proposal_generator.rpn_head.conv.weight,
%_wrapped_model.proposal_generator.rpn_head.conv.bias)
  %498 = Relu(%497)
  %499 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%498,
%_wrapped_model.proposal_generator.rpn_head.objectness_logits.weight,
%_wrapped_model.proposal_generator.rpn_head.objectness_logits.bias)
  %500 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%498,
%_wrapped_model.proposal_generator.rpn_head.anchor_deltas.weight,
%_wrapped_model.proposal_generator.rpn_head.anchor_deltas.bias)
  %501 = Conv[dilations = [1, 1], group = 1, kernel_shape = [3, 3], pads = [1,
1, 1, 1], strides = [1, 1]](%491,
%_wrapped_model.proposal_generator.rpn_head.conv.weight,
%_wrapped_model.proposal_generator.rpn_head.conv.bias)
  %502 = Relu(%501)
  %503 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%502,
%_wrapped_model.proposal_generator.rpn_head.objectness_logits.weight,
%_wrapped_model.proposal_generator.rpn_head.objectness_logits.bias)
  %504 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%502,
%_wrapped_model.proposal_generator.rpn_head.anchor_deltas.weight,
%_wrapped_model.proposal_generator.rpn_head.anchor_deltas.bias)
  %505 = Conv[dilations = [1, 1], group = 1, kernel_shape = [3, 3], pads = [1,
1, 1, 1], strides = [1, 1]](%487,
%_wrapped_model.proposal_generator.rpn_head.conv.weight,
%_wrapped_model.proposal_generator.rpn_head.conv.bias)
  %506 = Relu(%505)
  %507 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%506,
%_wrapped_model.proposal_generator.rpn_head.objectness_logits.weight,
%_wrapped_model.proposal_generator.rpn_head.objectness_logits.bias)
  %508 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%506,
%_wrapped_model.proposal_generator.rpn_head.anchor_deltas.weight,
%_wrapped_model.proposal_generator.rpn_head.anchor_deltas.bias)
  %509 = Conv[dilations = [1, 1], group = 1, kernel_shape = [3, 3], pads = [1,
1, 1, 1], strides = [1, 1]](%483,
%_wrapped_model.proposal_generator.rpn_head.conv.weight,
%_wrapped_model.proposal_generator.rpn_head.conv.bias)
  %510 = Relu(%509)
  %511 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%510,
%_wrapped_model.proposal_generator.rpn_head.objectness_logits.weight,
%_wrapped_model.proposal_generator.rpn_head.objectness_logits.bias)
```

```
  %512 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%510,
%_wrapped_model.proposal_generator.rpn_head.anchor_deltas.weight,
%_wrapped_model.proposal_generator.rpn_head.anchor_deltas.bias)
  %513 = Conv[dilations = [1, 1], group = 1, kernel_shape = [3, 3], pads = [1,
1, 1, 1], strides = [1, 1]](%496,
%_wrapped_model.proposal_generator.rpn_head.conv.weight,
%_wrapped_model.proposal_generator.rpn_head.conv.bias)
  %514 = Relu(%513)
  %515 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%514,
%_wrapped_model.proposal_generator.rpn_head.objectness_logits.weight,
%_wrapped_model.proposal_generator.rpn_head.objectness_logits.bias)
  %516 = Conv[dilations = [1, 1], group = 1, kernel_shape = [1, 1], pads = [0,
0, 0, 0], strides = [1, 1]](%514,
%_wrapped_model.proposal_generator.rpn_head.anchor_deltas.weight,
%_wrapped_model.proposal_generator.rpn_head.anchor_deltas.bias)
  %517, %518 = GenerateProposals[angle_bound_hi = 180, angle_bound_lo = -180,
angle_bound_on = 1, clip_angle_thresh = 1, legacy_plus_one = 0, min_size = 0,
nms_thresh = 0.699999988079071, post_nms_topN = 1000, pre_nms_topN = 1000,
spatial_scale = 0.25](%499, %500, %303,
%_wrapped_model.proposal_generator.anchor_generator.cell_anchors.0)
  %519, %520 = GenerateProposals[angle_bound_hi = 180, angle_bound_lo = -180,
angle_bound_on = 1, clip_angle_thresh = 1, legacy_plus_one = 0, min_size = 0,
nms_thresh = 0.699999988079071, post_nms_topN = 1000, pre_nms_topN = 1000,
spatial_scale = 0.125](%503, %504, %303,
%_wrapped_model.proposal_generator.anchor_generator.cell_anchors.1)
  %521, %522 = GenerateProposals[angle_bound_hi = 180, angle_bound_lo = -180,
angle_bound_on = 1, clip_angle_thresh = 1, legacy_plus_one = 0, min_size = 0,
nms_thresh = 0.699999988079071, post_nms_topN = 1000, pre_nms_topN = 1000,
spatial_scale = 0.0625](%507, %508, %303,
%_wrapped_model.proposal_generator.anchor_generator.cell_anchors.2)
  %523, %524 = GenerateProposals[angle_bound_hi = 180, angle_bound_lo = -180,
angle_bound_on = 1, clip_angle_thresh = 1, legacy_plus_one = 0, min_size = 0,
nms_thresh = 0.699999988079071, post_nms_topN = 1000, pre_nms_topN = 1000,
spatial_scale = 0.03125](%511, %512, %303,
%_wrapped_model.proposal_generator.anchor_generator.cell_anchors.3)
  %525, %526 = GenerateProposals[angle_bound_hi = 180, angle_bound_lo = -180,
angle_bound_on = 1, clip_angle_thresh = 1, legacy_plus_one = 0, min_size = 0,
nms_thresh = 0.699999988079071, post_nms_topN = 1000, pre_nms_topN = 1000,
spatial_scale = 0.015625](%515, %516, %303,
%_wrapped_model.proposal_generator.anchor_generator.cell_anchors.4)
  %rpn_rois = CollectRpnProposals[rpn_max_level = 6, rpn_min_level = 2,
rpn_post_nms_topN = 1000](%517, %519, %521, %523, %525, %518, %520, %522, %524,
%526)
  %528, %529, %530, %531, %532 = DistributeFpnProposals[legacy_plus_one = 0,
roi_canonical_level = 4, roi_canonical_scale = 224, roi_max_level = 5,
roi_min_level = 2](%rpn_rois)
```

```
  %533 = RoIAlign[aligned = 1, order = 'NCHW', pooled_h = 7, pooled_w = 7,
sampling_ratio = 0, spatial_scale = 0.25](%495, %528)
  %534 = RoIAlign[aligned = 1, order = 'NCHW', pooled_h = 7, pooled_w = 7,
sampling_ratio = 0, spatial_scale = 0.125](%491, %529)
  %535 = RoIAlign[aligned = 1, order = 'NCHW', pooled_h = 7, pooled_w = 7,
sampling_ratio = 0, spatial_scale = 0.0625](%487, %530)
  %roi_feat_fpn = RoIAlign[aligned = 1, order = 'NCHW', pooled_h = 7, pooled_w =
7, sampling_ratio = 0, spatial_scale = 0.03125](%483, %531)
  %537 = Concat[axis = 0](%533, %534, %535, %roi_feat_fpn)
  %538 = BatchPermutation(%537, %532)
  %539 = Flatten[axis = 1](%538)
  %540 = Gemm[alpha = 1, beta = 1, transB = 1](%539,
%_wrapped_model.roi_heads.box_head.fc1.weight,
%_wrapped_model.roi_heads.box_head.fc1.bias)
  %541 = Relu(%540)
  %542 = Gemm[alpha = 1, beta = 1, transB = 1](%541,
%_wrapped_model.roi_heads.box_head.fc2.weight,
%_wrapped_model.roi_heads.box_head.fc2.bias)
  %543 = Relu(%542)
  %544 = Gemm[alpha = 1, beta = 1, transB = 1](%543,
%_wrapped_model.roi_heads.box_predictor.cls_score.weight,
%_wrapped_model.roi_heads.box_predictor.cls_score.bias)
  %545 = Gemm[alpha = 1, beta = 1, transB = 1](%543,
%_wrapped_model.roi_heads.box_predictor.bbox_pred.weight,
%_wrapped_model.roi_heads.box_predictor.bbox_pred.bias)
  %546 = Softmax[axis = 1](%544)
  %547 = Concat[axis = 0](%rpn_rois)
  %548, %549 = BBoxTransform[angle_bound_hi = 180, angle_bound_lo = -180,
angle_bound_on = 1, apply_scale = 1, clip_angle_thresh = 1, legacy_plus_one = 0,
rotated = 0, weights = [10, 10, 5, 5]](%547, %545, %303)
  %roi_score_nms, %roi_bbox_nms.1, %roi_class_nms, %roi_batch_splits_nms,
%roi_keeps_nms, %roi_keeps_size_nms = BoxWithNMSLimit[cls_agnostic_bbox_reg = 0,
detections_per_im = 100, input_boxes_include_bg_cls = 0, legacy_plus_one = 0,
nms = 0.5, output_classes_include_bg_cls = 0, rotated = 0, score_thresh = 0.5,
soft_nms_enabled = 0, soft_nms_method = 'linear', soft_nms_min_score_thres =
0.00100000004749745, soft_nms_sigma = 0.5](%546, %548, %549)
  %556 = AliasWithName[is_backward = 0, name = 'class_nms'](%roi_class_nms)
  %557 = AliasWithName[is_backward = 0, name = 'score_nms'](%roi_score_nms)
  %roi_bbox_nms = AliasWithName[is_backward = 0, name =
'bbox_nms'](%roi_bbox_nms.1)
  return %roi_bbox_nms, %557, %556
}
```

[12]:
```
from detectron2.export.caffe2_modeling import␣
 ↪convert_batched_inputs_to_c2_format, META_ARCH_CAFFE2_EXPORT_TYPE_MAP

# image preparation from `DefaultPredictor`
```

```
# https://github.com/facebookresearch/detectron2/blob/master/detectron2/engine/
 ↪defaults.py#L214
original_image = cv2.imread('datasets/coco/val2017/000000023937.jpg')

aug = T.ResizeShortestEdge(
    [cfg.INPUT.MIN_SIZE_TEST, cfg.INPUT.MIN_SIZE_TEST], cfg.INPUT.MAX_SIZE_TEST
)

height, width = original_image.shape[:2]
image = aug.get_transform(original_image).apply_image(original_image)
image = torch.as_tensor(image.astype("float32").transpose(2, 0, 1))

inputs = {"image": image, "height": height, "width": width}
(image, img_info) = convert_batched_inputs_to_c2_format([inputs], 32, "cpu")
```

[13]:
```
print(model.graph.input[0])
print(model.graph.input[1])

# two inputs are needed `data` wich is the image and `img_info` wich represents␣
 ↪the shape of the image
W = {
    model.graph.input[0].name: image.data.numpy(),
    model.graph.input[1].name: img_info.data.numpy()
}
```

```
name: "data"
type {
  tensor_type {
    elem_type: 2
    shape {
      dim {
        dim_value: 1
      }
      dim {
        dim_value: 3
      }
      dim {
        dim_value: 800
      }
      dim {
        dim_value: 1216
      }
    }
  }
}

name: "im_info"
```

```
type {
  tensor_type {
    elem_type: 1
    shape {
      dim {
        dim_value: 1
      }
      dim {
        dim_value: 3
      }
    }
  }
}
```

[14]:
```
# run onnx inference
rep = backend.prepare(model)
raw_onnx_outputs = rep.run(W)
```

```
/home/users/fassbinb/.local/lib/python3.8/site-
packages/caffe2/python/onnx/backend.py:690: UserWarning: Unrecognized operator
set org.pytorch._caffe2
  warnings.warn("Unrecognized operator set {}".format(imp.domain))
```

[15]:
```
onnx_outputs = {
    "bbox_nms": torch.tensor(raw_onnx_outputs.roi_bbox_nms),
    "score_nms": torch.tensor(raw_onnx_outputs._1),
    "class_nms": torch.tensor(raw_onnx_outputs._2)
}
```

[16]:
```
# convert output to detectron2 compatible output
from detectron2.modeling import meta_arch
from detectron2.export.caffe2_modeling import assemble_rcnn_outputs_by_name

image_sizes = [[int(im[0]), int(im[1])] for im in img_info]
results = assemble_rcnn_outputs_by_name(image_sizes, onnx_outputs)

# replace with the used model
outputs = getattr(meta_arch, cfg.MODEL.META_ARCHITECTURE)._postprocess(results,␣
 ↪[inputs], image_sizes)
```

[17]:
```
v = Visualizer(original_image[:, :, ::-1], MetadataCatalog.get(cfg.DATASETS.
 ↪TRAIN[0]), scale=1.2)
out = v.draw_instance_predictions(outputs[0]["instances"].to("cpu"))
cv2_imshow(out.get_image()[:, :, ::-1])
```


sheep 94%
cow 94%
sheep 52%
0
100
200
300
400
500
0
100
200
300
400
500
600
700