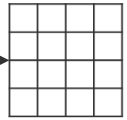
Faster R-CNN算法改进：多个检测层

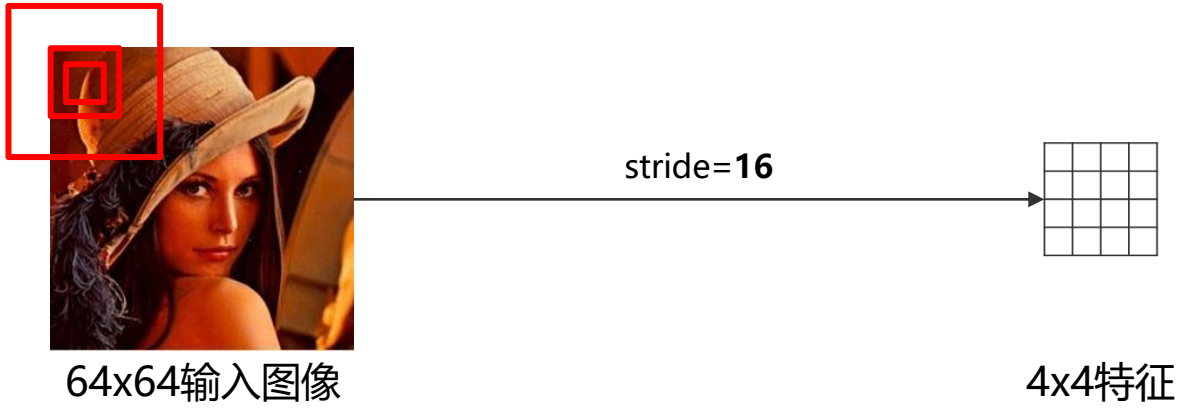
64x64输入图像

stride=**16**

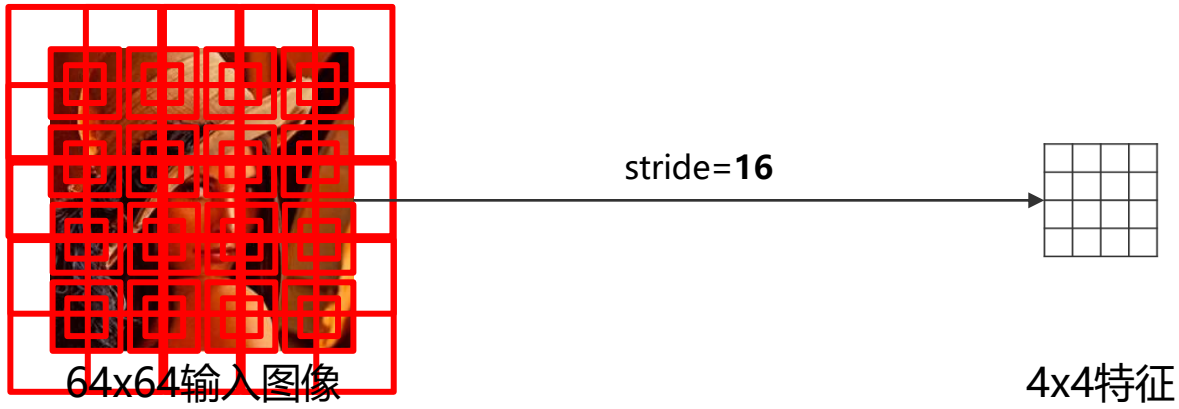


4x4特征

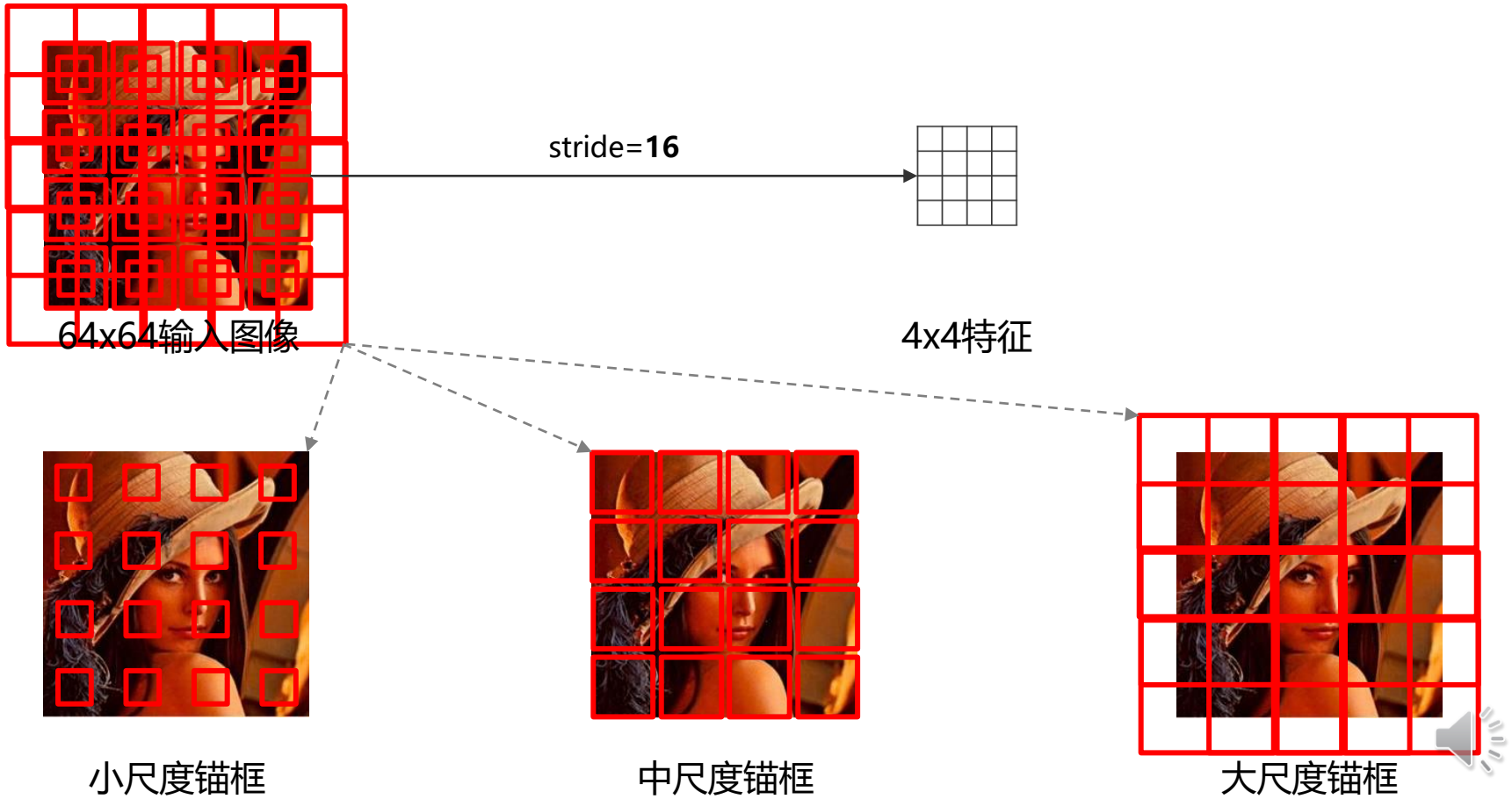
Faster R-CNN算法改进：多个检测层



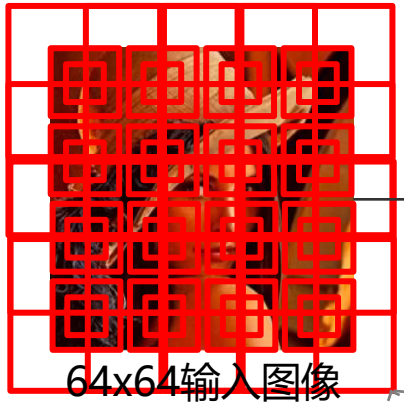
Faster R-CNN算法改进：多个检测层



Faster R-CNN算法改进：多个检测层

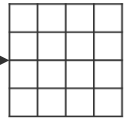


Faster R-CNN算法改进：多个检测层



64x64输入图像

stride=**16**



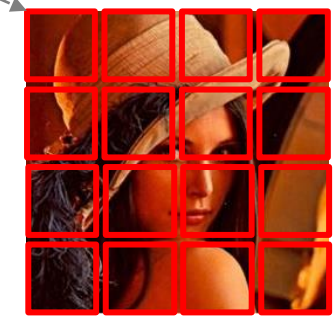
4x4特征

单个检测层的问题

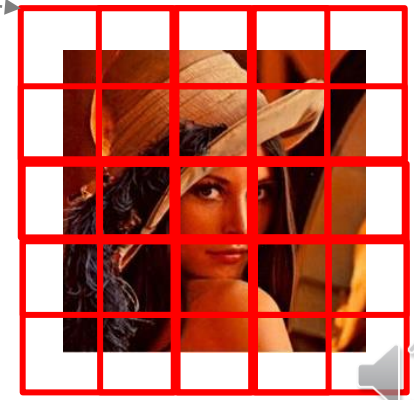
① **小尺度锚框铺设太稀疏**

② **大尺度锚框铺设太稠密**

小尺度锚框



中尺度锚框

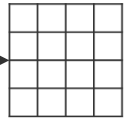


大尺度锚框

Faster R-CNN算法改进：多个检测层

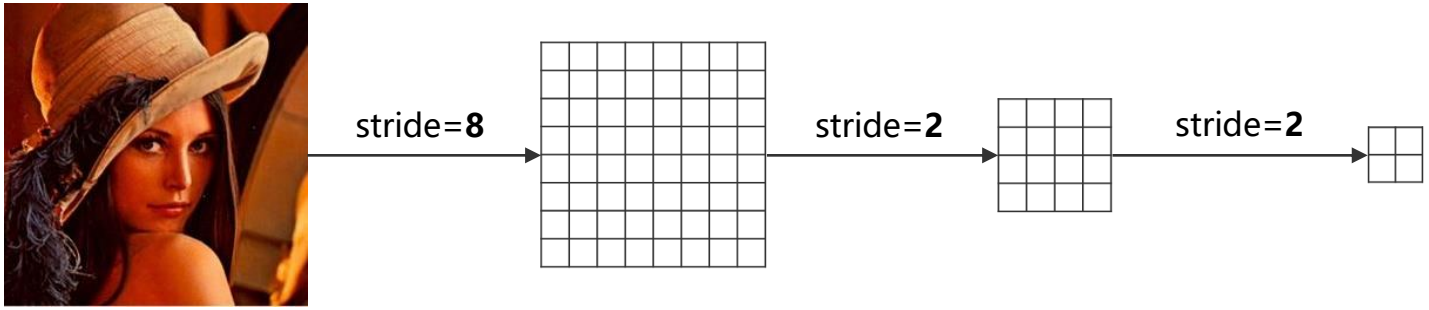
64x64输入图像

stride=**16**

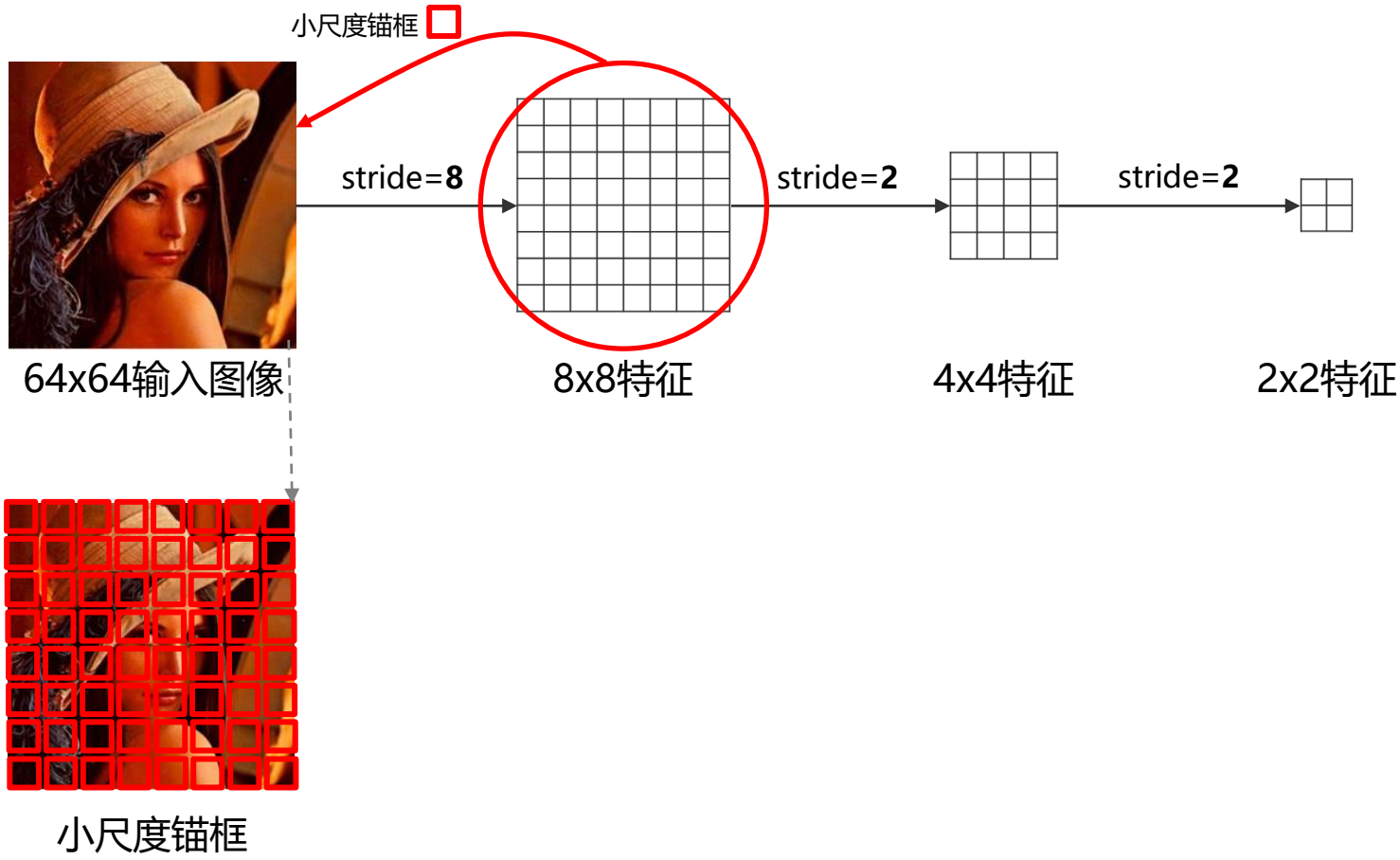


4x4特征

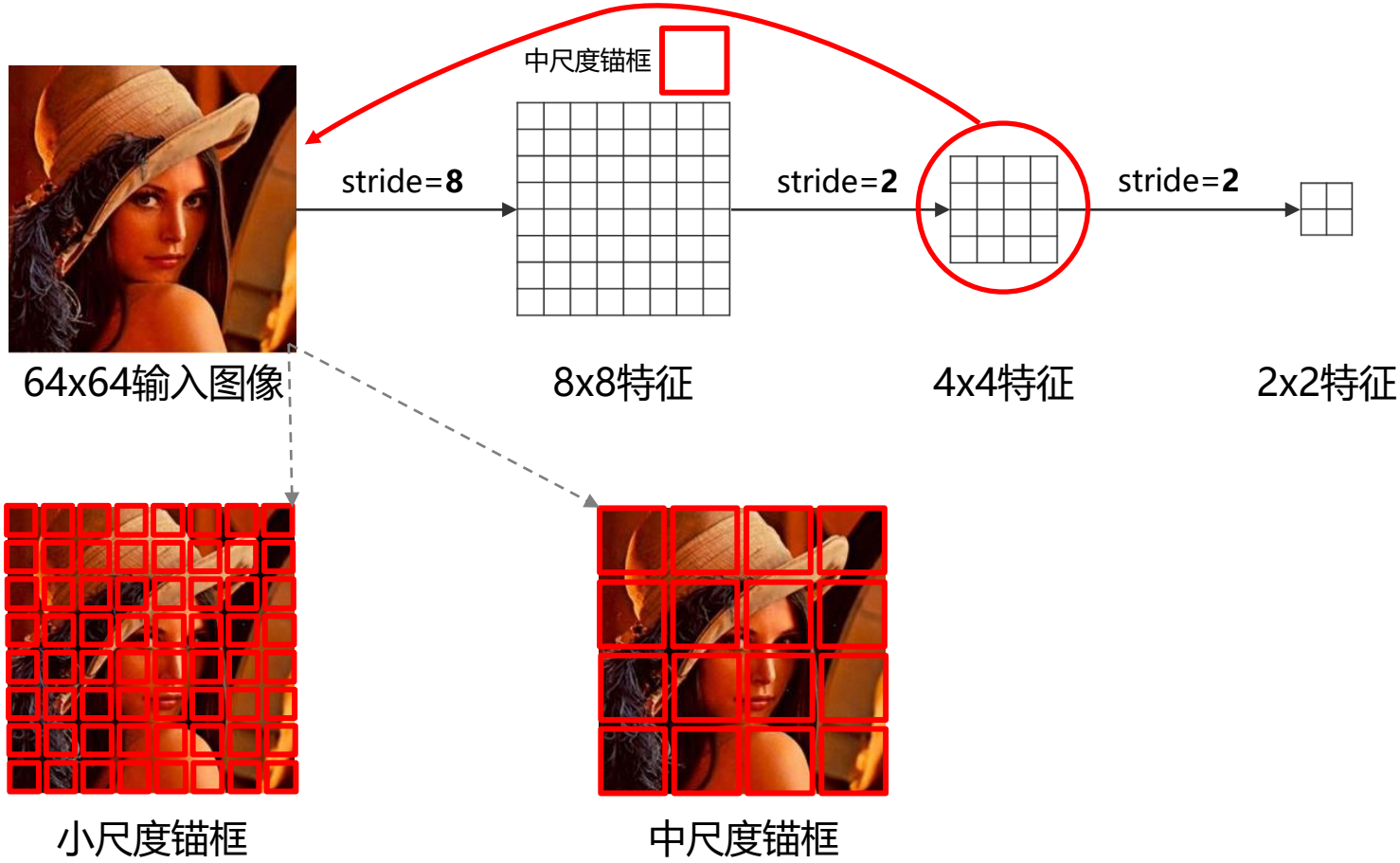
Faster R-CNN算法改进：多个检测层



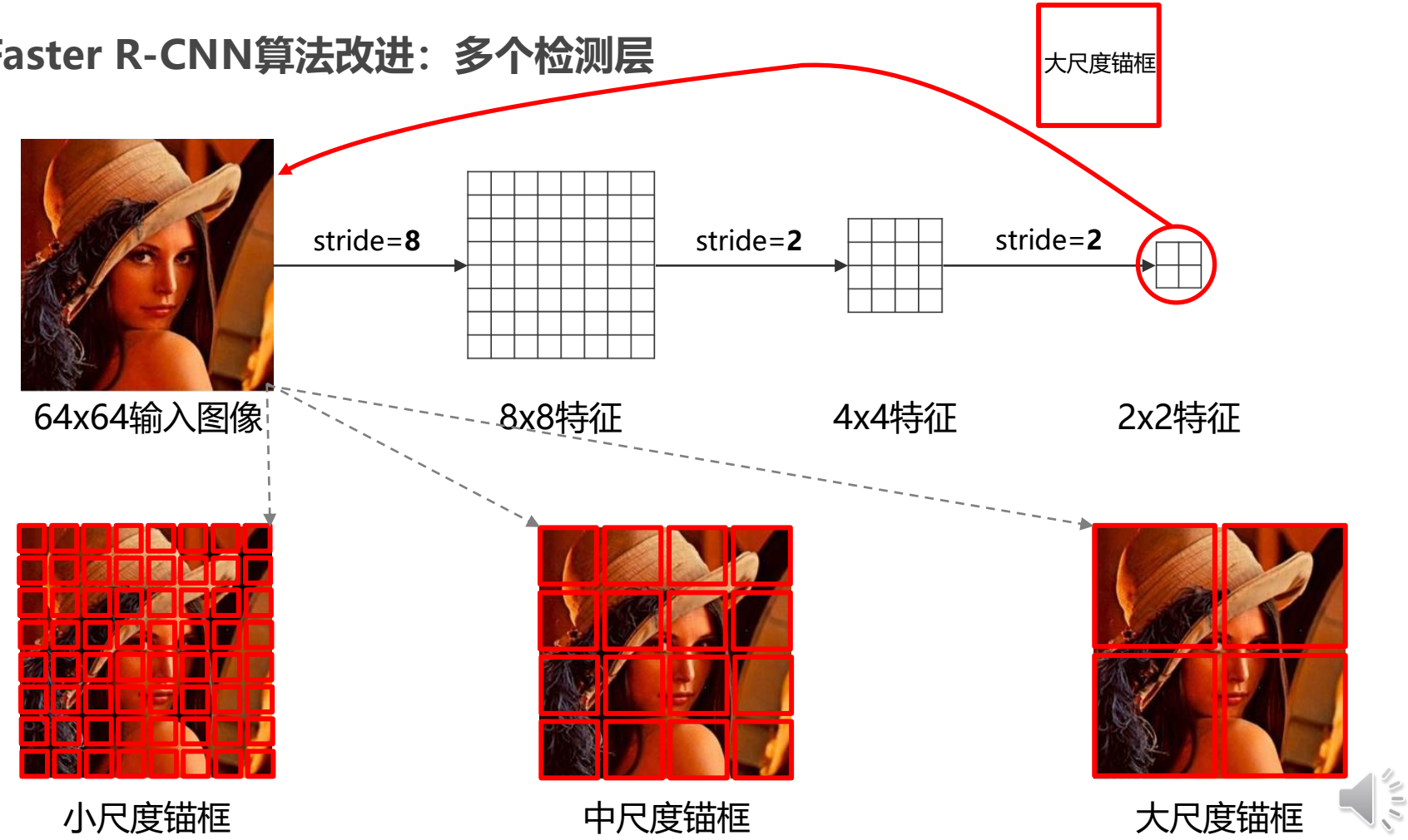
Faster R-CNN算法改进：多个检测层



Faster R-CNN算法改进：多个检测层



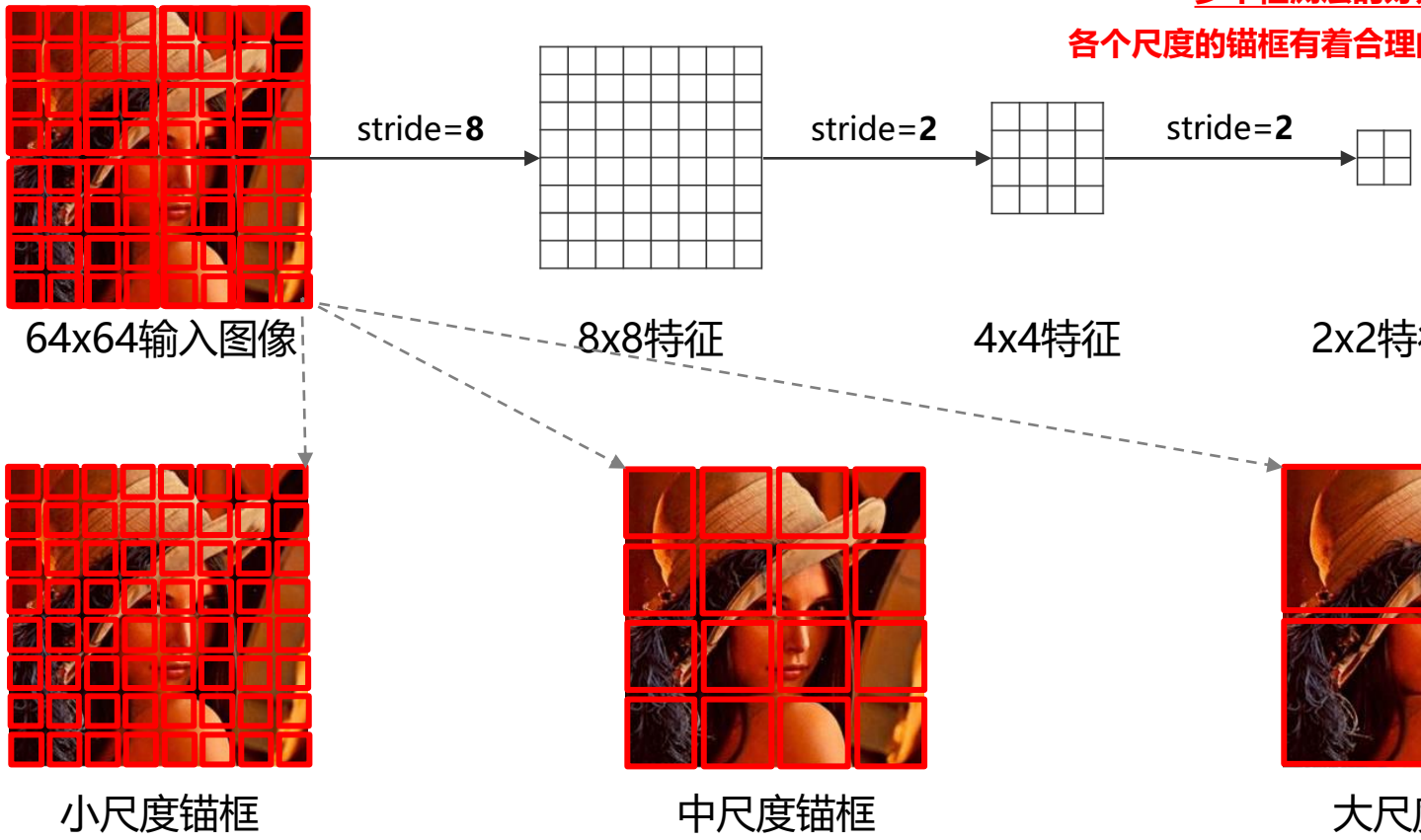
Faster R-CNN算法改进：多个检测层



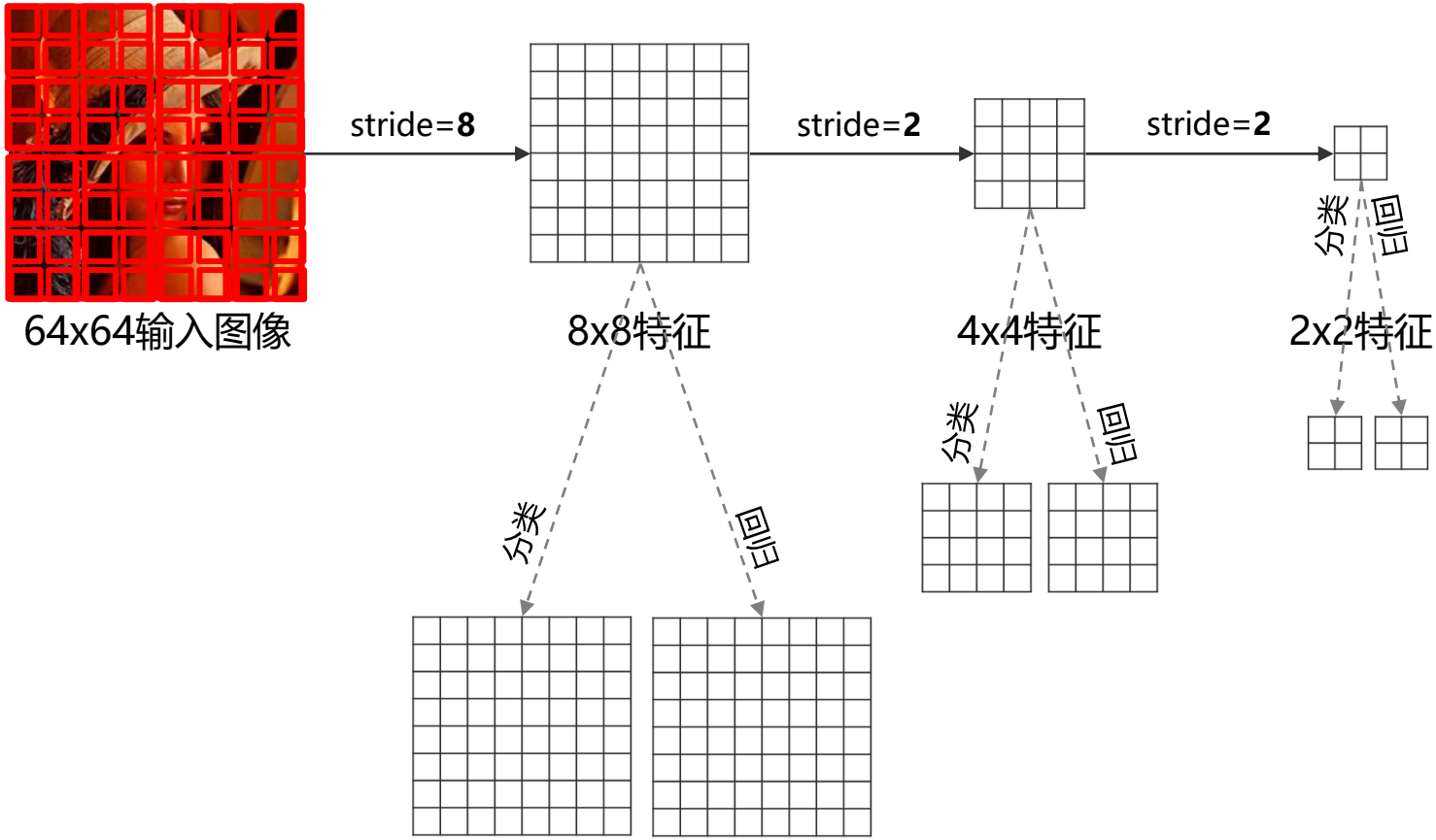
Faster R-CNN算法改进：多个检测层

多个检测层的好处

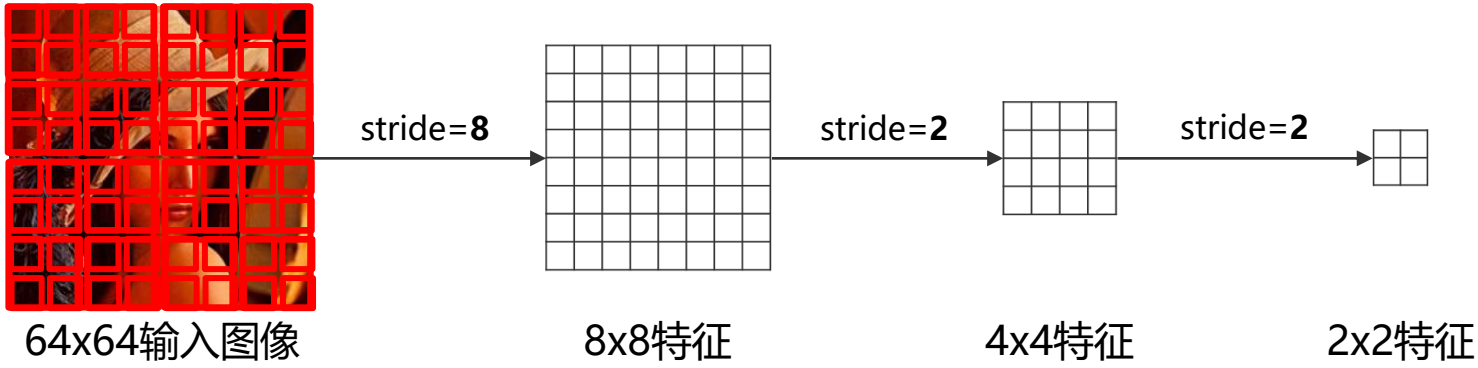
各个尺度的锚框有着合理的铺设间隔



Faster R-CNN算法改进：多个检测层

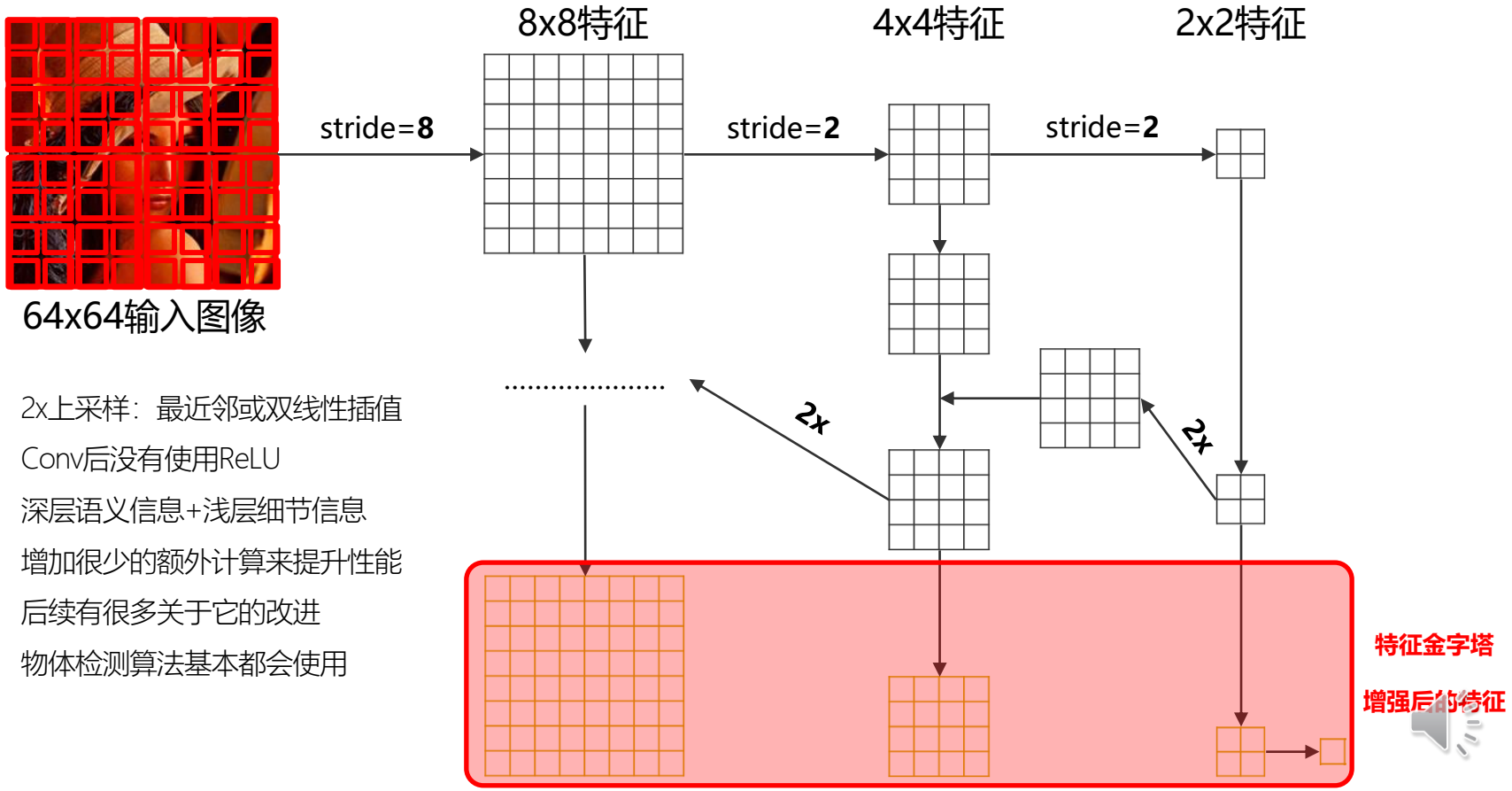


Faster R-CNN算法改进：特征金字塔 (FPN, Feature Pyramid Network)



- 物体检测 = 物体分类 + 物体定位
- 物体分类：网络层数较深时，分辨率较小，语义信息比较丰富，利于分类
- 物体回归：网络层数较浅时，分辨率较大，细节信息比较丰富，利于回归
- 存在问题：现有分类网络中，某层的特征不能同时满足这两点，即语义和细节都丰富
- 解决方案：特征金字塔从后到前对特征进行融合，让每层的特征都具备丰富的语义和细节

Faster R-CNN算法改进：特征金字塔 (FPN, Feature Pyramid Network)



- 2x上采样：最近邻或双线性插值
- Conv后没有使用ReLU
- 深层语义信息+浅层细节信息
- 增加很少的额外计算来提升性能
- 后续有很多关于它的改进
- 物体检测算法基本都会使用

Faster R-CNN代码架构解读：配置文件

faster_rcnn_R_50_FPN_1x.yaml

```yaml
_BASE_: "../Base-RCNN-FPN.yaml"
MODEL:
  WEIGHTS: "detectron2://ImageNetPretrained/MSRA/R-50.pkl"
  MASK_ON: False
  RESNETS:
    DEPTH: 50
```

- faster_rcnn_R_50_FPN_1x.yaml

Faster R-CNN代码架构解读：配置文件

faster_rcnn_R_50_FPN_1x.yaml

```yaml
_BASE_: "../Base-RCNN-FPN.yaml"
MODEL:
  WEIGHTS: "detectron2://ImageNetPretrained/MSRA/R-50.pkl"
  MASK_ON: False
  RESNETS:
    DEPTH: 50
```

- faster_rcnn_R_50_FPN_1x.yaml
- 基础网络是ImageNet预训练的ResNet50，加上FPN

Faster R-CNN代码架构解读：配置文件

faster_rcnn_R_50_FPN_1x.yaml

```yaml
_BASE_: "../Base-RCNN-FPN.yaml"
MODEL:
  WEIGHTS: "detectron2://ImageNetPretrained/MSRA/R-50.pkl"
  MASK_ON: False
  RESNETS:
    DEPTH: 50
```

Base-RCNN-FPN.yaml

```yaml
MODEL:
  META_ARCHITECTURE: "GeneralizedRCNN"
  BACKBONE:
    NAME: "build_resnet_fpn_backbone"
  RESNETS:
    OUT_FEATURES: ["res2", "res3", "res4", "res5"]
  FPN:
    IN_FEATURES: ["res2", "res3", "res4", "res5"]
  ANCHOR_GENERATOR:
    SIZES: [[32], [64], [128], [256], [512]]  # One size for each in featu
    ASPECT_RATIOS: [[0.5, 1.0, 2.0]]  # Three aspect ratios (same for all
  RPN:
    IN_FEATURES: ["p2", "p3", "p4", "p5", "p6"]
    PRE_NMS_TOPK_TRAIN: 2000  # Per FPN level
    PRE_NMS_TOPK_TEST: 1000  # Per FPN level
    # Detectron1 uses 2000 proposals per-batch,
    # (See "modeling/rpn/rpn_outputs.py" for details of this legacy issue)
    # which is approximately 1000 proposals per-image since the default ba
    POST_NMS_TOPK_TRAIN: 1000
    POST_NMS_TOPK_TEST: 1000
  ROI_HEADS:
    NAME: "StandardROIHeads"
    IN_FEATURES: ["p2", "p3", "p4", "p5"]
  ROI_BOX_HEAD:
    NAME: "FastRCNNConvFCHead"
    NUM_FC: 2
    POOLER_RESOLUTION: 7
  ROI_MASK_HEAD:
    NAME: "MaskRCNNConvUpsampleHead"
    NUM_CONV: 4
    POOLER_RESOLUTION: 14
DATASETS:
  TRAIN: ("coco_2017_train",)
  TEST: ("coco_2017_val",)
SOLVER:
  IMS_PER_BATCH: 16
  BASE_LR: 0.02
  STEPS: (60000, 80000)
  MAX_ITER: 90000
INPUT:
  MIN_SIZE_TRAIN: (640, 672, 704, 736, 768, 800)
VERSION: 2
```

- faster_rcnn_R_50_FPN_1x.yaml
- 基础网络是ImageNet预训练的ResNet50，加上FPN

Faster R-CNN代码架构解读：配置文件

faster_rcnn_R_50_FPN_1x.yaml

```yaml
_BASE_: "../Base-RCNN-FPN.yaml"
MODEL:
  WEIGHTS: "detectron2://ImageNetPretrained/MSRA/R-50.pkl"
  MASK_ON: False
  RESNETS:
    DEPTH: 50
```

Base-RCNN-FPN.yaml

```yaml
MODEL:
  META_ARCHITECTURE: "GeneralizedRCNN"
  BACKBONE:
    NAME: "build_resnet_fpn_backbone"
  RESNETS:
    OUT_FEATURES: ["res2", "res3", "res4", "res5"]
  FPN:
    IN_FEATURES: ["res2", "res3", "res4", "res5"]
  ANCHOR_GENERATOR:
    SIZES: [[32], [64], [128], [256], [512]]  # One size for each in featu
    ASPECT_RATIOS: [[0.5, 1.0, 2.0]]  # Three aspect ratios (same for all
  RPN:
    IN_FEATURES: ["p2", "p3", "p4", "p5", "p6"]
    PRE_NMS_TOPK_TRAIN: 2000  # Per FPN level
    PRE_NMS_TOPK_TEST: 1000  # Per FPN level
    # Detectron1 uses 2000 proposals per-batch,
    # (See "modeling/rpn/rpn_outputs.py" for details of this legacy issue)
    # which is approximately 1000 proposals per-image since the default ba
    POST_NMS_TOPK_TRAIN: 1000
    POST_NMS_TOPK_TEST: 1000
  ROI_HEADS:
    NAME: "StandardROIHeads"
    IN_FEATURES: ["p2", "p3", "p4", "p5"]
  ROI_BOX_HEAD:
    NAME: "FastRCNNConvFCHead"
    NUM_FC: 2
    POOLER_RESOLUTION: 7
  ROI_MASK_HEAD:
    NAME: "MaskRCNNConvUpsampleHead"
    NUM_CONV: 4
    POOLER_RESOLUTION: 14
DATASETS:
  TRAIN: ("coco_2017_train",)
  TEST: ("coco_2017_val",)
SOLVER:
  IMS_PER_BATCH: 16
  BASE_LR: 0.02
  STEPS: (60000, 80000)
  MAX_ITER: 90000
INPUT:
  MIN_SIZE_TRAIN: (640, 672, 704, 736, 768, 800)
VERSION: 2
```

- faster_rcnn_R_50_FPN_1x.yaml
- 基础网络是ImageNet预训练的ResNet50，加上FPN
- 初始检测层：res2、res3、res4、res5
- 最终检测层：P2、 P3、 P4、 P5、 P6

Faster R-CNN代码架构解读：配置文件

faster_rcnn_R_50_FPN_1x.yaml

```yaml
_BASE_: "../Base-RCNN-FPN.yaml"
MODEL:
  WEIGHTS: "detectron2://ImageNetPretrained/MSRA/R-50.pkl"
  MASK_ON: False
  RESNETS:
    DEPTH: 50
```

Base-RCNN-FPN.yaml

```yaml
MODEL:
  META_ARCHITECTURE: "GeneralizedRCNN"
  BACKBONE:
    NAME: "build_resnet_fpn_backbone"
  RESNETS:
    OUT_FEATURES: ["res2", "res3", "res4", "res5"]
  FPN:
    IN_FEATURES: ["res2", "res3", "res4", "res5"]
  ANCHOR_GENERATOR:
    SIZES: [[32], [64], [128], [256], [512]]  # One size for each in featu
    ASPECT_RATIOS: [[0.5, 1.0, 2.0]]  # Three aspect ratios (same for all
  RPN:
    IN_FEATURES: ["p2", "p3", "p4", "p5", "p6"]
    PRE_NMS_TOPK_TRAIN: 2000  # Per FPN level
    PRE_NMS_TOPK_TEST: 1000  # Per FPN level
    # Detectron1 uses 2000 proposals per-batch,
    # (See "modeling/rpn/rpn_outputs.py" for details of this legacy issue)
    # which is approximately 1000 proposals per-image since the default ba
    POST_NMS_TOPK_TRAIN: 1000
    POST_NMS_TOPK_TEST: 1000
  ROI_HEADS:
    NAME: "StandardROIHeads"
    IN_FEATURES: ["p2", "p3", "p4", "p5"]
  ROI_BOX_HEAD:
    NAME: "FastRCNNConvFCHead"
    NUM_FC: 2
    POOLER_RESOLUTION: 7
  ROI_MASK_HEAD:
    NAME: "MaskRCNNConvUpsampleHead"
    NUM_CONV: 4
    POOLER_RESOLUTION: 14
DATASETS:
  TRAIN: ("coco_2017_train",)
  TEST: ("coco_2017_val",)
SOLVER:
  IMS_PER_BATCH: 16
  BASE_LR: 0.02
  STEPS: (60000, 80000)
  MAX_ITER: 90000
INPUT:
  MIN_SIZE_TRAIN: (640, 672, 704, 736, 768, 800)
VERSION: 2
```

- faster_rcnn_R_50_FPN_1x.yaml
- 基础网络是ImageNet预训练的ResNet50，加上FPN
- 初始检测层：res2、res3、res4、res5
- 最终检测层：P2、 P3、 P4、 P5、 P6
- 检测层stride：4、 8、 16、 32、 64
- 锚框大小： 32、 64、 128、 256、 512
- 锚框比例：每层都是[0.5, 1.0, 2.0]

Faster R-CNN代码架构解读：配置文件

faster_rcnn_R_50_FPN_1x.yaml

```yaml
_BASE_: "../Base-RCNN-FPN.yaml"
MODEL:
  WEIGHTS: "detectron2://ImageNetPretrained/MSRA/R-50.pkl"
  MASK_ON: False
  RESNETS:
    DEPTH: 50
```

- faster_rcnn_R_50_FPN_1x.yaml
- 基础网络是ImageNet预训练的ResNet50，加上FPN
- 初始检测层：res2、res3、res4、res5
- 最终检测层：P2、 P3、 P4、 P5、 P6
- 检测层stride：4、 8、 16、 32、 64
- 锚框大小： 32、 64、 128、 256、 512
- 锚框比例：每层都是[0.5, 1.0, 2.0]

Base-RCNN-FPN.yaml

```yaml
MODEL:
  META_ARCHITECTURE: "GeneralizedRCNN"
  BACKBONE:
    NAME: "build_resnet_fpn_backbone"
  RESNETS:
    OUT_FEATURES: ["res2", "res3", "res4", "res5"]
  FPN:
    IN_FEATURES: ["res2", "res3", "res4", "res5"]
  ANCHOR_GENERATOR:
    SIZES: [[32], [64], [128], [256], [512]]  # One size for each in featu
    ASPECT_RATIOS: [[0.5, 1.0, 2.0]]  # Three aspect ratios (same for all
  RPN:
    IN_FEATURES: ["p2", "p3", "p4", "p5", "p6"]
    PRE_NMS_TOPK_TRAIN: 2000  # Per FPN level
    PRE_NMS_TOPK_TEST: 1000  # Per FPN level
    # Detectron1 uses 2000 proposals per-batch,
    # (See "modeling/rpn/rpn_outputs.py" for details
    # which is approximately 1000 proposals per-image
    POST_NMS_TOPK_TRAIN: 1000
    POST_NMS_TOPK_TEST: 1000
  ROI_HEADS:
    NAME: "StandardROIHeads"
    IN_FEATURES: ["p2", "p3", "p4", "p5"]
  ROI_BOX_HEAD:
    NAME: "FastRCNNConvFCHead"
    NUM_FC: 2
    POOLER_RESOLUTION: 7
  ROI_MASK_HEAD:
    NAME: "MaskRCNNConvUpsampleHead"
    NUM_CONV: 4
    POOLER_RESOLUTION: 14
DATASETS:
  TRAIN: ("coco_2017_train",)
  TEST: ("coco_2017_val",)
SOLVER:
  IMS_PER_BATCH: 16
  BASE_LR: 0.02
  STEPS: (60000, 80000)
  MAX_ITER: 90000
INPUT:
  MIN_SIZE_TRAIN: (640, 672, 704, 736, 768, 800)
VERSION: 2
```

RPN生成候选区域的参数

Fast RCNN的参数

训练和测试的数据

训练参数

多尺度训练

Faster R-CNN代码架构解读：算法流程

构建

- 模型构建
- 优化器构建
- 数据构建

训练

- 数据处理
- 特征提取
- RPN训练
- RPN生成候选区域
- Fast R-CNN训练

测试

- 数据处理
- 特征提取
- RPN生成候选区域
- Fast R-CNN输出结果

Faster R-CNN代码架构解读：算法流程

构建

- 模型构建
- 优化器构建
- 数据构建

Faster R-CNN代码架构解读：构建

```
model = self.build_model(cfg)
optimizer = self.build_optimizer(cfg, model)
data_loader = self.build_train_loader(cfg)
```

Faster R-CNN代码架构解读：模型构建

```
model = self.build_model(cfg)
optimizer = self.build_optimizer(cfg, model)
data_loader = self.build_train_loader(cfg)
```

```
self.backbone = build_backbone(cfg)
self.proposal_generator = build_proposal_generator(cfg, self.backbone.output_shape())
self.roi_heads = build_roi_heads(cfg, self.backbone.output_shape())
```

Faster R-CNN代码架构解读：模型构建

```
model = self.build_model(cfg)
optimizer = self.build_optimizer(cfg, model)
data_loader = self.build_train_loader(cfg)
```

```
self.backbone = build_backbone(cfg)
self.proposal_generator = build_proposal_generator(cfg, self.backbone.output_shape())
self.roi_heads = build_roi_heads(cfg, self.backbone.output_shape())
```

```
def build_resnet_fpn_backbone(cfg, input_shape: ShapeSpec):
    """
    Args:
        cfg: a detectron2 CfgNode

    Returns:
        backbone (Backbone): backbone module, must be a subclass of :class:`Backbone`
    """
    bottom_up = build_resnet_backbone(cfg, input_shape)
    in_features = cfg.MODEL.FPN.IN_FEATURES
    out_channels = cfg.MODEL.FPN.OUT_CHANNELS
    backbone = FPN(
        bottom_up=bottom_up,
        in_features=in_features,
        out_channels=out_channels,
        norm=cfg.MODEL.FPN.NORM,
        top_block=LastLevelMaxPool(),
        fuse_type=cfg.MODEL.FPN.FUSE_TYPE,
    )
    return backbone
```

ResNet构建

FPN构建

Faster R-CNN代码架构解读：模型构建

```
model = self.build_model(cfg)
optimizer = self.build_optimizer(cfg, model)
data_loader = self.build_train_loader(cfg)
```

```
self.backbone = build_backbone(cfg)
self.proposal_generator = build_proposal_generator(cfg, self.backbone.output_shape())
self.roi_heads = build_roi_heads(cfg, self.backbone.output_shape())
```

```
def build_resnet_fpn_backbone(cfg, input_shape: ShapeSpec):
    """
    Args:
        cfg: a detectron2 CfgNode

    Returns:
        backbone (Backbone): backbone module, must be a subclass of :class:`Backbone`.
    """
    bottom_up = build_resnet_backbone(cfg, input_shape)
    in_features = cfg.MODEL.FPN.IN_FEATURES
    out_channels = cfg.MODEL.FPN.OUT_CHANNELS
    backbone = FPN(
        bottom_up=bottom_up,
        in_features=in_features,
        out_channels=out_channels,
        norm=cfg.MODEL.FPN.NORM,
        top_block=LastLevelMaxPool(),
        fuse_type=cfg.MODEL.FPN.FUSE_TYPE,
    )
    return backbone
```

```
self.anchor_generator = build_anchor_generator(
    cfg, [input_shape[f] for f in self.in_features]
)
self.box2box_transform = Box2BoxTransform(weights=cfg.MODEL.RPN.BBOX_REG_WEIGHTS)
self.anchor_matcher = Matcher(
    cfg.MODEL.RPN.IOU_THRESHOLDS, cfg.MODEL.RPN.IOU_LABELS, allow_low_quality_matches=True
)
self.rpn_head = build_rpn_head(cfg, [input_shape[f] for f in self.in_features])
```

锚框的构建

锚框变换的构建

锚框匹配的构建

锚框预测的构建

Faster R-CNN代码架构解读：模型构建

```
model = self.build_model(cfg)
optimizer = self.build_optimizer(cfg, model)
data_loader = self.build_train_loader(cfg)
```

```
self.backbone = build_backbone(cfg)
self.proposal_generator = build_proposal_generator(cfg, self.backbone.output_shape())
self.roi_heads = build_roi_heads(cfg, self.backbone.output_shape())
```

```
def build_resnet_fpn_backbone(cfg, input_shape: ShapeSpec):
    """
    Args:
        cfg: a detectron2 CfgNode

    Returns:
        backbone (Backbone): backbone module, must be a subclass of :class:`Backbone`
    """
    bottom_up = build_resnet_backbone(cfg, input_shape)
    in_features = cfg.MODEL.FPN.IN_FEATURES
    out_channels = cfg.MODEL.FPN.OUT_CHANNELS
    backbone = FPN(
        bottom_up=bottom_up,
        in_features=in_features,
        out_channels=out_channels,
        norm=cfg.MODEL.FPN.NORM,
        top_block=LastLevelMaxPool(),
        fuse_type=cfg.MODEL.FPN.FUSE_TYPE,
    )
    return backbone
```

```
self.anchor_generator = build_anchor_generator(
    cfg, [input_shape[f] for f in self.in_features]
)
self.box2box_transform = Box2BoxTransform(weights=cfg.MODEL.RPN.BBOX_REG_WEIGHTS)
self.anchor_matcher = Matcher(
    cfg.MODEL.RPN.IOU_THRESHOLDS, cfg.MODEL.RPN.IOU_LABELS, allow_low_quality_matches=True
)
self.rpn_head = build_rpn_head(cfg, [input_shape[f] for f in self.in_features])
```

```
self.box_pooler = box_pooler
self.box_head = box_head
self.box_predictor = box_predictor
```

RoIPooling构建

特征增强网络构建

预测模块构建

Faster R-CNN代码架构解读：优化器构建

```
model = self.build_model(cfg)
optimizer = self.build_optimizer(cfg, model)
data_loader = self.build_train_loader(cfg)
```

```
self.scheduler = self.build_lr_scheduler(cfg, optimizer)
# Assume no other objects need to be checkpointed.
# We can later make it checkpoint the stateful hooks
self.checkpointer = DetectionCheckpointer(
    # Assume you want to save checkpoints together with logs/statistics
    model,
    cfg.OUTPUT_DIR,
    optimizer=optimizer,
    scheduler=self.scheduler,
)
self.start_iter = 0
self.max_iter = cfg.SOLVER.MAX_ITER
self.cfg = cfg
```

学习率策略

训练模型保存

训练迭代次数

Faster R-CNN代码架构解读：数据构建

```
model = self.build_model(cfg)
optimizer = self.build_optimizer(cfg, model)
data_loader = self.build_train_loader(cfg)
```

```
dataset_dicts = get_detection_dataset_dicts(
    cfg.DATASETS.TRAIN,
    filter_empty=cfg.DATALOADER.FILTER_EMPTY_ANNOTATIONS,
    min_keypoints=cfg.MODEL.ROI_KEYPOINT_HEAD.MIN_KEYPOINTS_PER_IMAGE
    if cfg.MODEL.KEYPOINT_ON
    else 0,
    proposal_files=cfg.DATASETS.PROPOSAL_FILES_TRAIN if cfg.MODEL.LOAD_PROPOSALS else None,
)
dataset = DatasetFromList(dataset_dicts, copy=False)

if mapper is None:
    mapper = DatasetMapper(cfg, True)
dataset = MapDataset(dataset, mapper)

sampler_name = cfg.DATALOADER.SAMPLER_TRAIN
logger = logging.getLogger(__name__)
logger.info("Using training sampler {}".format(sampler_name))
# TODO avoid if-else?
if sampler_name == "TrainingSampler":
    sampler = samplers.TrainingSampler(len(dataset))
elif sampler_name == "RepeatFactorTrainingSampler":
    sampler = samplers.RepeatFactorTrainingSampler(
        dataset_dicts, cfg.DATALOADER.REPEAT_THRESHOLD
    )
else:
    raise ValueError("Unknown training sampler: {}".format(sampler_name))
return build_batch_data_loader(
    dataset,
    sampler,
    cfg.SOLVER.IMS_PER_BATCH,
    aspect_ratio_grouping=cfg.DATALOADER.ASPECT_RATIO_GROUPING,
    num_workers=cfg.DATALOADER.NUM_WORKERS,
```

Faster R-CNN代码架构解读：算法流程

构建

- 模型构建
- 优化器构建
- 数据构建

训练

- 数据处理
- 特征提取
- RPN训练
- RPN生成候选区域
- Fast R-CNN训练

Faster R-CNN代码架构解读：数据处理

读图像

```
image = utils.read_image(dataset_dict["file_name"], format=self.img_format)
```

图像变换

（缩放、水平翻转）

```
image, transforms = T.apply_transform_gens(self.tfm_gens, image)
```

标注变换

```
utils.transform_instance_annotations(
    obj, transforms, image_shape, keypoint_hflip_indices=self.keypoint_hflip_indices
```

过滤标注

```
dataset_dict["instances"] = utils.filter_empty_instances(instances)
```

图像归一化

```
images = [(x - self.pixel_mean) / self.pixel_std for x in images]
```

图像组batch

```
images = ImageList.from_tensors(images, self.backbone.size_divisibility)
```

Faster R-CNN代码架构解读：特征提取

```
bottom_up_features = self.bottom_up(x)
x = [bottom_up_features[f] for f in self.in_features[::-1]]
results = []
prev_features = self.lateral_convs[0](x[0])
results.append(self.output_convs[0](prev_features))
for features, lateral_conv, output_conv in zip(
    x[1:], self.lateral_convs[1:], self.output_convs[1:]
):
    top_down_features = F.interpolate(prev_features, scale_factor=2, mode="nearest")
    lateral_features = lateral_conv(features)
    prev_features = lateral_features + top_down_features
    if self._fuse_type == "avg":
        prev_features /= 2
    results.insert(0, output_conv(prev_features))

if self.top_block is not None:
    top_block_in_feature = bottom_up_features.get(self.top_block.in_feature, None)
    if top_block_in_feature is None:
        top_block_in_feature = results[self._out_features.index(self.top_block.in_feature)]
    results.extend(self.top_block(top_block_in_feature))
assert len(self._out_features) == len(results)
return dict(zip(self._out_features, results))
```

ResNet特征提取

FPN特征提取

Faster R-CNN代码架构解读：RPN训练和候选区域生成

```
anchors = self.anchor_generator(features)
```

生成锚框

```
pred_objectness_logits, pred_anchor_deltas = self.rpn_head(features)
```

预测锚框的
类别和偏移

```
gt_labels, gt_boxes = self.label_and_sample_anchors(anchors, gt_instances)
```

计算锚框类别和
偏移的真值

```
losses = self.losses(
    anchors, pred_objectness_logits, gt_labels, pred_anchor_deltas, gt_boxes
)
```

计算分类和
回归的误差损失

```
proposals = self.predict_proposals(
    anchors, pred_objectness_logits, pred_anchor_deltas, images.image_sizes
)
```

RPN生成
候选区域

Faster R-CNN代码架构解读：Fast R-CNN训练

```
proposals = add_ground_truth_to_proposals(gt_boxes, proposals)
```

把真实标注框
加入候选区域

```
proposals = self.label_and_sample_proposals(proposals, targets)
```

候选区域的分类和回
归的真值计算

```
box_features = self.box_pooler(features, [x.proposal_boxes for x in proposals])
```

RoIPooling

```
box_features = self.box_head(box_features)
```

候选区域
特征增强

```
predictions = self.box_predictor(box_features)
```

候选区域分类和
回归的预测

```
F.cross_entropy(self.pred_class_logits, self.gt_classes, reduction="mean")
```

分类损失函数

```
loss_box_reg = smooth_l1_loss(
    self.pred_proposal_deltas[fg_inds[:, None], gt_class_cols],
    gt_proposal_deltas[fg_inds],
    self.smooth_l1_beta,
    reduction="sum",
)
```

回归损失函数

请观看演示视频

--config-file
./configs/COCO-Detection/faster_rcnn_R_50_FPN_1x.yaml
--num-gpus
1
SOLVER.IMS_PER_BATCH
1
INPUT.MIN_SIZE_TRAIN
(400,)
DATASETS.TRAIN
('coco_2017_val',)
DATALOADER.NUM_WORKERS
0

课程作业

- 单步调试Faster R-CNN代码

1. 利用配好的detectron2物体检测平台，使用PyCharm软件，单步调试如下配置(https://github.com/facebookresearch/detectron2/blob/master/configs/COCO-Detection/faster_rcnn_R_50_FPN_1x.yaml)的Faster R-CNN代码
2. 把Faster R-CNN中每个细节与代码对应上，真正弄懂Faster R-CNN的整个流程
3. (可选) 如果硬件条件允许，可以使用8卡GPU训一个模型，看精度是否与官方一致

* 大家一定多花些时间仔细调试Faster R-CNN的代码，有不懂的可以在群里问或者查阅相关资料，Faster R-CNN的代码、原理、细节等都搞明白之后，后续的物体检测算法都很容易上手

感谢各位聆听!

Thanks for Listening