Global wheat detection

Cùng ngâm cứu 2 model FASTERRCNN và EFFICIENTDET xem thử

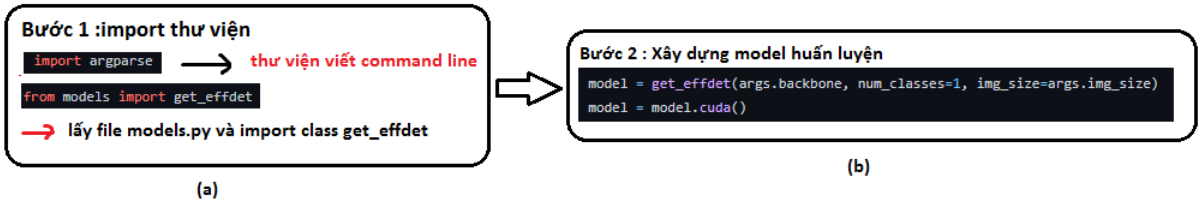
cách anh Dũng top 1 xây dựng model như thế nào nhé ?

1.EFFICIENTDET

Link: https://github.com/dungnb1333/global-wheat-dection-2020/blob/main/effdet_train.py

effdet_train.py

Đầu tiên, ta xét đầu ra của model



Hình 1 : thư viện (a) và đầu ra mong muốn của model (b)

Tiếp theo, cùng đi sâu vào thư viện models.py xem nó có gì nhé !! ^^

```
from effdet import get_efficientdet_config, EfficientDet, DetBenchEval, DetBenchTrain
from effdet.efficientdet import HeadNet
```

```
def get_effdet(backbone, num_classes=1, img_size=512,
               mode='train', pretrained=True, pretrained_backbone=False):
 >>...  (1)
if mode == 'train':
    return DetBenchTrain(model, config)
else:
    return DetBenchEval(model, config)
```

Đầu ra mong muốn

>>(1)

```
if pretrained:
    pretrained_backbone = False
if backbone == 'ed0':
    config = get_efficientdet_config('tf_efficientdet_d0')
    checkpoint = torch.load('effdet-pretrained/tf_efficientdet_d0-d92fd44f.pth')
elif backbone == 'ed1':
    config = get_efficientdet_config('tf_efficientdet_d1')
    checkpoint = torch.load('effdet-pretrained/tf_efficientdet_d1-4c7ebaf2.pth')
elif backbone == 'ed2':
    config = get_efficientdet_config('tf_efficientdet_d2')
    checkpoint = torch.load('effdet-pretrained/tf_efficientdet_d2-cb4ce77d.pth')
elif backbone == 'ed3':
    config = get_efficientdet_config('tf_efficientdet_d3')
    checkpoint = torch.load('effdet-pretrained/tf_efficientdet_d3-b0ea2cbc.pth')
elif backbone == 'ed4':
    config = get_efficientdet_config('tf_efficientdet_d4')
    checkpoint = torch.load('effdet-pretrained/tf_efficientdet_d4-5b370b7a.pth')
elif backbone == 'ed5':
    config = get_efficientdet_config('tf_efficientdet_d5')
    checkpoint = torch.load('effdet-pretrained/tf_efficientdet_d5-ef44aea8.pth')
elif backbone == 'ed6':
    config = get_efficientdet_config('tf_efficientdet_d6')
    checkpoint = torch.load('effdet-pretrained/tf_efficientdet_d6-51cb0132.pth')
elif backbone == 'ed7':
    config = get_efficientdet_config('tf_efficientdet_d7')
    checkpoint = torch.load('effdet-pretrained/tf_efficientdet_d7-f05bf714.pth')
```

```
else:
    raise ValueError("BACKBONE!!!")
model = EfficientDet(config, pretrained_backbone=pretrained_backbone)
if pretrained:
    model.load_state_dict(checkpoint)
    del checkpoint
    gc.collect()
config.num_classes = num_classes
config.image_size = img_size
model.class_net = HeadNet(config, num_outputs=config.num_classes, norm_kwargs=dict(eps=.001, momentum=.01))
```

Tiếp tục đi vào hai hàm đầu ra xem ảnh xây dựng như thế nào nào.

```python
class DetBenchEval(nn.Module):
    def __init__(self, model, config):
        super(DetBenchEval, self).__init__()
        self.config = config
        self.model = model
        self.anchors = Anchors(
            config.min_level, config.max_level,
            config.num_scales, config.aspect_ratios,
            config.anchor_scale, config.image_size)

    def update(self, image_size):
        self.config.image_size = image_size
        self.anchors = Anchors(
            self.config.min_level, self.config.max_level,
            self.config.num_scales, self.config.aspect_ratios,
            self.config.anchor_scale, self.config.image_size)
        self.anchors = self.anchors.cuda()

    def forward(self, x, image_scales):
        class_out, box_out = self.model(x)
        class_out, box_out, indices, classes = _post_process(self.config, class_out, box_out)

        batch_detections = []
        # FIXME we may be able to do this as a batch with some tensor reshaping/indexing, PR welcome
        for i in range(x.shape[0]):
            detections = generate_detections(
                class_out[i], box_out[i], self.anchors.boxes, indices[i], classes[i], image_scales[i])
            batch_detections.append(detections)
        return torch.stack(batch_detections, dim=0)
```

https://github.com/dungnb1333/global-wheat-dection-2020/blob/main/effdet/bench.py

```python
def _post_process(config, cls_outputs, box_outputs):
    """Selects top-k predictions.

    Post-proc code adapted from Tensorflow version at: https://github.com/google/automl/tree/master/efficientdet
    and optimized for PyTorch.

    Args:
        config: a parameter dictionary that includes `min_level`, `max_level`,  `batch_size`, and `num_classes`.

        cls_outputs: an OrderDict with keys representing levels and values
            representing logits in [batch_size, height, width, num_anchors].

        box_outputs: an OrderDict with keys representing levels and values
            representing box regression targets in [batch_size, height, width, num_anchors * 4].
    """
    batch_size = cls_outputs[0].shape[0]
    cls_outputs_all = torch.cat([
        cls_outputs[level].permute(0, 2, 3, 1).reshape([batch_size, -1, config.num_classes])
        for level in range(config.num_levels)], 1)

    box_outputs_all = torch.cat([
        box_outputs[level].permute(0, 2, 3, 1).reshape([batch_size, -1, 4])
        for level in range(config.num_levels)], 1)

    _, cls_topk_indices_all = torch.topk(cls_outputs_all.reshape(batch_size, -1), dim=1, k=MAX_DETECTION_POINTS)
    indices_all = cls_topk_indices_all / config.num_classes
    classes_all = cls_topk_indices_all % config.num_classes
```

```python
box_outputs_all_after_topk = torch.gather(
    box_outputs_all, 1, indices_all.unsqueeze(2).expand(-1, -1, 4))

cls_outputs_all_after_topk = torch.gather(
    cls_outputs_all, 1, indices_all.unsqueeze(2).expand(-1, -1, config.num_classes))
cls_outputs_all_after_topk = torch.gather(
    cls_outputs_all_after_topk, 2, classes_all.unsqueeze(2))

return cls_outputs_all_after_topk, box_outputs_all_after_topk, indices_all, classes_all
```

```python
class DetBenchTrain(nn.Module):
    def __init__(self, model, config):
        super(DetBenchTrain, self).__init__()
        self.config = config
        self.model = model
        anchors = Anchors(
            config.min_level, config.max_level,
            config.num_scales, config.aspect_ratios,
            config.anchor_scale, config.image_size)
        self.anchor_labeler = AnchorLabeler(anchors, config.num_classes, match_threshold=0.5)
        self.loss_fn = DetectionLoss(self.config)

    def forward(self, x, gt_boxes, gt_labels):
        class_out, box_out = self.model(x)

        cls_targets = []
        box_targets = []
        num_positives = []
        # FIXME this may be a bottleneck, would be faster if batched, or should be done in loader/dataset?
        for i in range(x.shape[0]):
            gt_class_out, gt_box_out, num_positive = self.anchor_labeler.label_anchors(gt_boxes[i], gt_labels[i])
            cls_targets.append(gt_class_out)
            box_targets.append(gt_box_out)
            num_positives.append(num_positive)

        return self.loss_fn(class_out, box_out, cls_targets, box_targets, num_positives)
```

Nhiều thư viện quá, mắc mệt !!!

```python
from .anchors import Anchors, AnchorLabeler, generate_detections, MAX_DETECTION_POINTS
from .loss import DetectionLoss
```

https://github.com/dungnb1333/global-wheat-dection-2020/blob/main/effdet/anchors.py