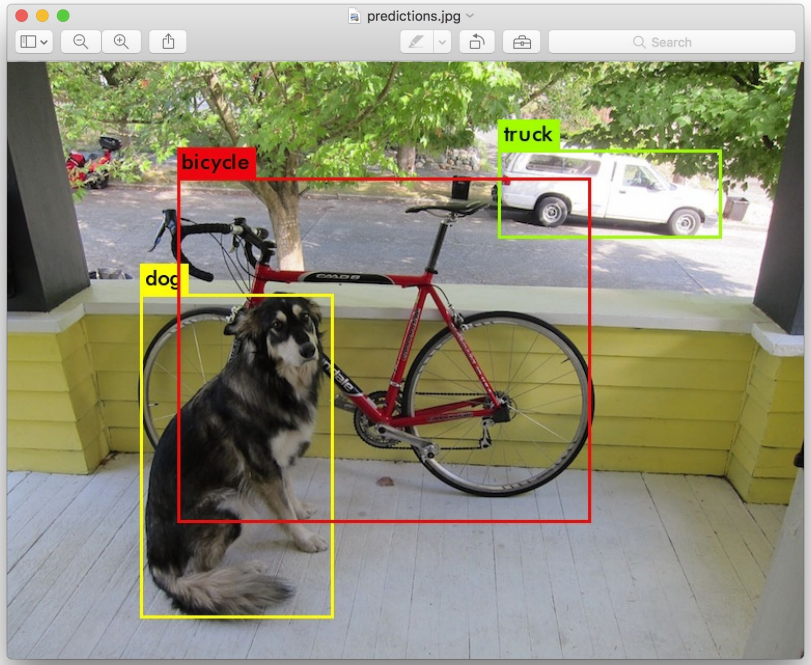
Faster R-CNN

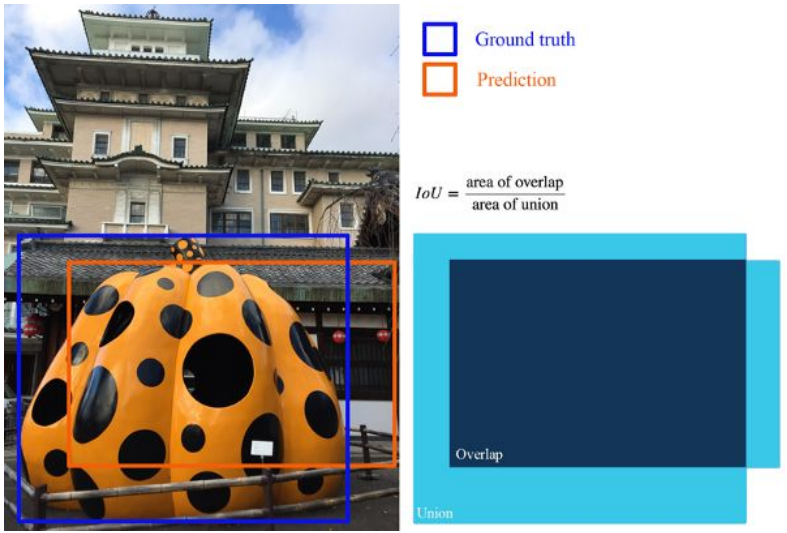
code approach

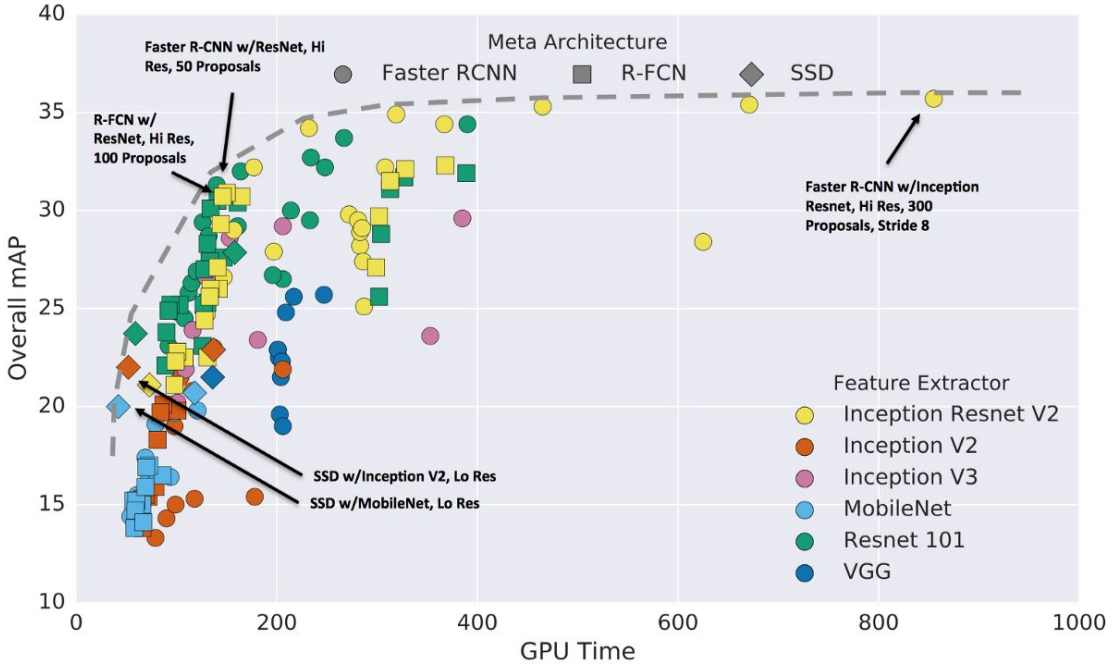
Evgeny Ponomarev

Object detection



$$\text{mAP} = \frac{1}{|Q_R|} \sum_{q \in Q_R} \text{AP}(q)$$



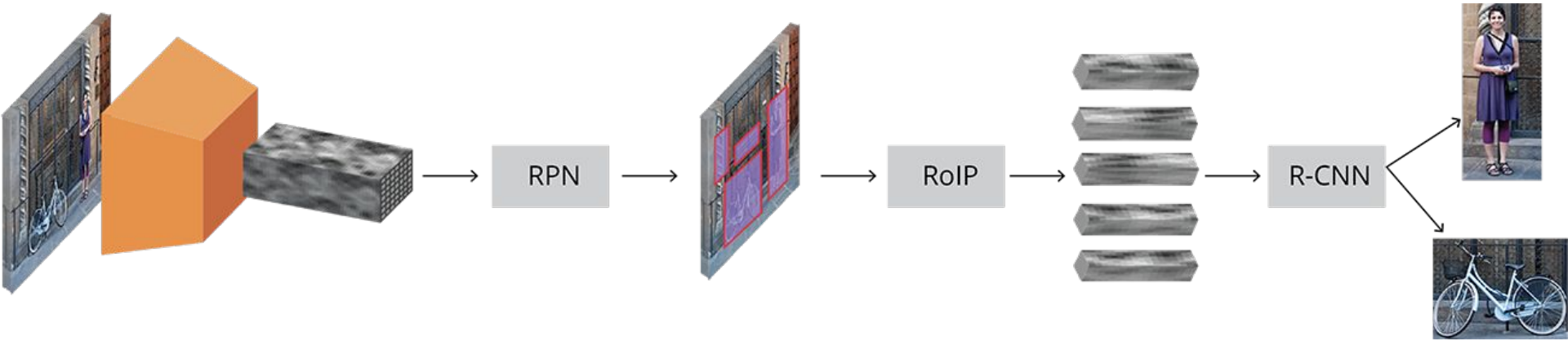


Speed vs accuracy trade-off. A higher mAP and a lower GPU Time is optimal. COCO 🐔 dataset

Architecture

It all starts with an image, from which we want to obtain:

- a list of bounding boxes.
- a label assigned to each bounding box.
- a probability for each label and bounding box.



```
GeneralizedRCNN(
  (backbone): Sequential(
    (body): ResNet(
      ... SOME RESNET LAYERS ...
    )
  )
  (rpn): RPNModule(
    (anchor_generator): AnchorGenerator(
      (cell_anchors): BufferList()
    )
    (head): RPNHead(
      (conv): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (cls_logits): Conv2d(1024, 9, kernel_size=(1, 1), stride=(1, 1))
      (bbox_pred): Conv2d(1024, 36, kernel_size=(1, 1), stride=(1, 1))
    )
    (box_selector_train): RPNPostProcessor()
    (box_selector_test): RPNPostProcessor()
  )
  (roi_heads): CombinedROIHeads(
    (box): ROIBoxHead(
      (feature_extractor): ResNet50Conv5ROIFeatureExtractor(
        (pooler): Pooler(
          (poolers): ModuleList(
            (0): ROIAlign(output_size=(14, 14), spatial_scale=0.0625, sampling_ratio=0)
          )
        )
        (head): ResNetHead(
          ... SOME RESNET LAYERS ...
        )
      )
      (predictor): FastRCNNPredictor(
        (avgpool): AvgPool2d(kernel_size=7, stride=7, padding=0)
        (cls_score): Linear(in_features=2048, out_features=21, bias=True)
        (bbox_pred): Linear(in_features=2048, out_features=84, bias=True)
      )
      (post_processor): PostProcessor()
    )
  )
)
```

```
GeneralizedRCNN(
  (backbone): Sequential(
    (body): ResNet(
      ... SOME RESNET LAYERS ...
    )
  )
  (rpn): RPNModule(
    (anchor_generator): AnchorGenerator(
      (cell_anchors): BufferList()
    )
    (head): RPNHead(
      (conv): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (cls_logits): Conv2d(1024, 9, kernel_size=(1, 1), stride=(1, 1))
      (bbox_pred): Conv2d(1024, 36, kernel_size=(1, 1), stride=(1, 1))
    )
    (box_selector_train): RPNPostProcessor()
    (box_selector_test): RPNPostProcessor()
  )
  (roi_heads): CombinedROIHeads(
    (box): ROIBoxHead(
      (feature_extractor): ResNet50Conv5ROIFeatureExtractor(
        (pooler): Pooler(
          (poolers): ModuleList(
            (0): ROIAlign(output_size=(14, 14), spatial_scale=0.0625, sampling_ratio=0)
          )
        )
        (head): ResNetHead(
          ... SOME RESNET LAYERS ...
        )
      )
      (predictor): FastRCNNPredictor(
        (avgpool): AvgPool2d(kernel_size=7, stride=7, padding=0)
        (cls_score): Linear(in_features=2048, out_features=21, bias=True)
        (bbox_pred): Linear(in_features=2048, out_features=84, bias=True)
      )
      (post_processor): PostProcessor()
    )
  )
)
```

1

2

3

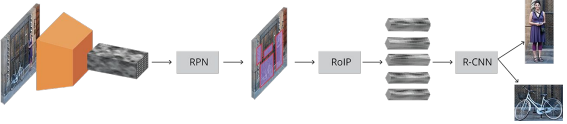
3.5

Whole model forward

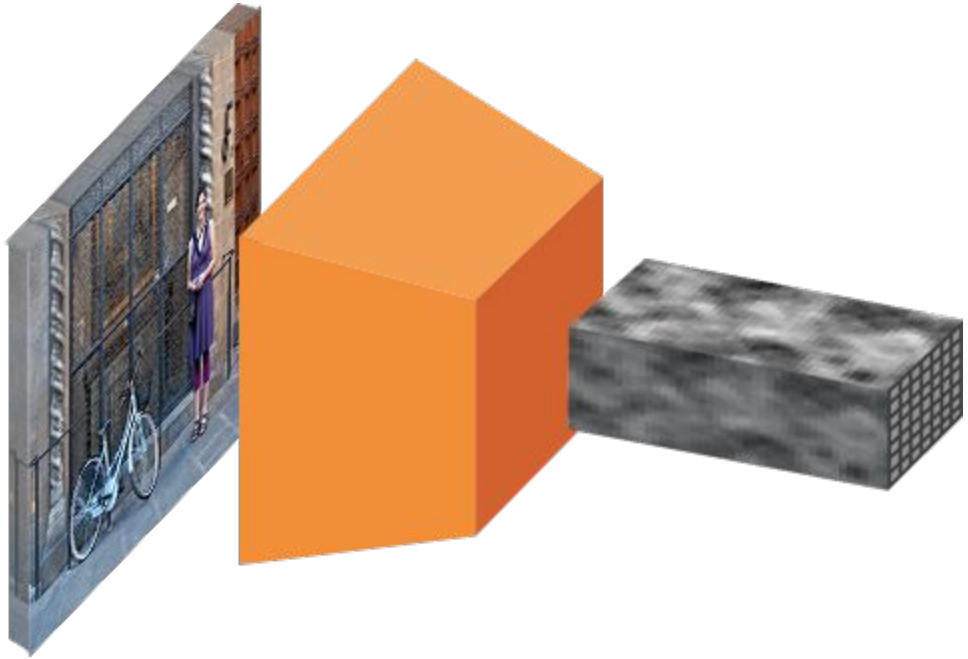
```python
def forward(self, images, targets=None):
    """
    Arguments:
        images (list[Tensor] or ImageList): images to be processed
        targets (list[BoxList]): ground-truth boxes present in the image (optional)

    Returns:
        result (list[BoxList] or dict[Tensor]): the output from the model.
            During training, it returns a dict[Tensor] which contains the losses.
            During testing, it returns list[BoxList] contains additional fields
            like `scores`, `labels` and `mask` (for Mask R-CNN models).

    """
    if self.training and targets is None:
        raise ValueError("In training mode, targets should be passed")
    images = to_image_list(images)
    features = self.backbone(images.tensors)
    proposals, proposal_losses = self.rpn(images, features, targets)
    if self.roi_heads:
        x, result, detector_losses = self.roi_heads(features, proposals, targets)
    else:
        # RPN-only models don't have roi_heads
        x = features
        result = proposals
        detector_losses = {}

    if self.training:
        losses = {}
        losses.update(detector_losses)
        losses.update(proposal_losses)
        return losses

    return result
```

```
  (backbone): Sequential(
    (body): ResNet(
        ... SOME RESNET LAYERS ...
    )
  )
  (rpn): RPNModule(
    (anchor_generator): AnchorGenerator(
      (cell_anchors): BufferList()
    )
    (head): RPNHead(
      (conv): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (cls_logits): Conv2d(1024, 9, kernel_size=(1, 1), stride=(1, 1))
      (bbox_pred): Conv2d(1024, 36, kernel_size=(1, 1), stride=(1, 1))
    )
    (box_selector_train): RPNPostProcessor()
    (box_selector_test): RPNPostProcessor()
  )
  (roi_heads): CombinedROIHeads(
    (box): ROIBoxHead(
      (feature_extractor): ResNet50Conv5ROIFeatureExtractor(
        (pooler): Pooler(
          (poolers): ModuleList(
            (0): ROIAlign(output_size=(14, 14), spatial_scale=0.0625, sampling_ratio=0)
          )
        )
        (head): ResNetHead(
          ... SOME RESNET LAYERS ...
        )
      )
      (predictor): FastRCNNPredictor(
        (avgpool): AvgPool2d(kernel_size=7, stride=7, padding=0)
        (cls_score): Linear(in_features=2048, out_features=21, bias=True)
        (bbox_pred): Linear(in_features=2048, out_features=84, bias=True)
      )
      (post_processor): PostProcessor()
    )
  )
)
```

```
(eneralizedRCNN(
  (backbone): Sequential(
    (body): ResNet(
      (stem): StemWithFixedBatchNorm(
        (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
        (bn1): FrozenBatchNorm2d()
      )
      (layer1): Sequential(
        (0): BottleneckWithFixedBatchNorm(
          (downsample): Sequential(
            (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): FrozenBatchNorm2d()
          )
          (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): FrozenBatchNorm2d()
          (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d()
          (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): FrozenBatchNorm2d()
        )
        (1): BottleneckWithFixedBatchNorm(...)
      )
      (layer2): Sequential(
        (0): BottleneckWithFixedBatchNorm(...)
        (1): BottleneckWithFixedBatchNorm(...)
        (2): BottleneckWithFixedBatchNorm(...)
        (3): BottleneckWithFixedBatchNorm(...)
      )
      (layer3): Sequential(
        (0): BottleneckWithFixedBatchNorm(...)
        (1): BottleneckWithFixedBatchNorm(...)
        (2): BottleneckWithFixedBatchNorm(...)
        (3): BottleneckWithFixedBatchNorm(...)
        (4): BottleneckWithFixedBatchNorm(...)
        (5): BottleneckWithFixedBatchNorm(...)
      )
    )
  )
  (rpn): RPNModule(
    (anchor_generator): AnchorGenerator(
      (cell_anchors): BufferList()
    )
```

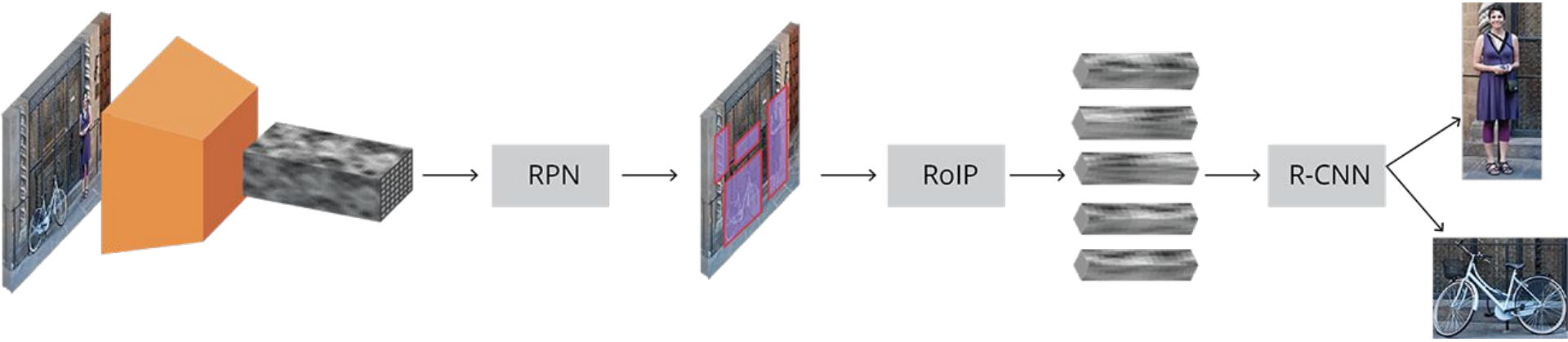


Backbone

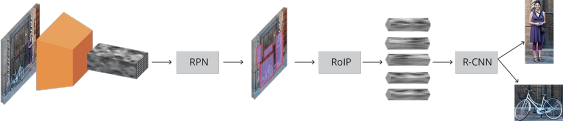


Input:
image batch
typ: [1, 3, 1066, 800]

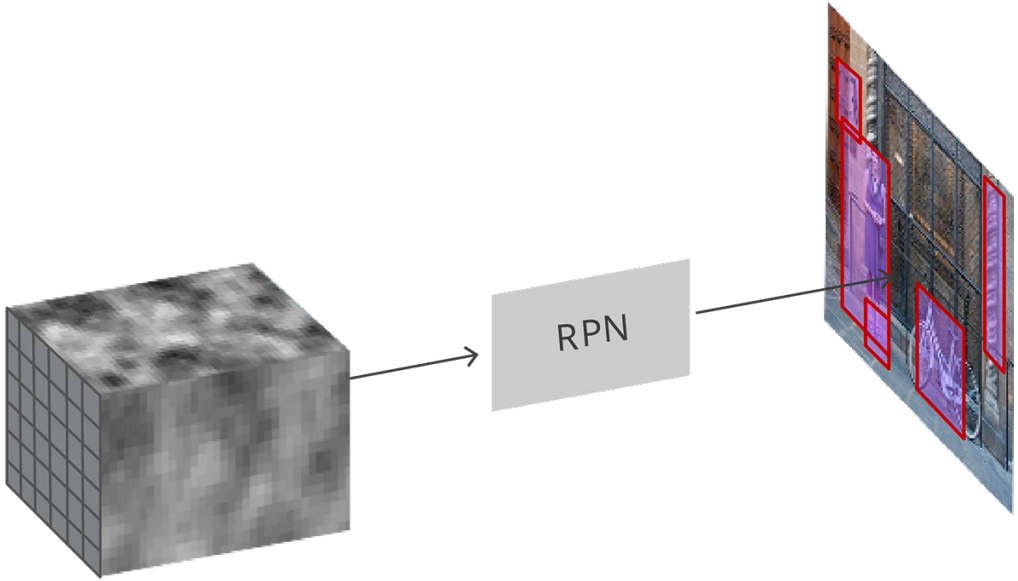
Output:
feature map
typ: [1, 1024, 67, 50]


RPN
RoIP
R-CNN

```
GeneralizedRCNN(
  (backbone): Sequential(
    (body): ResNet(
      ... SOME RESNET LAYERS ...
    )
  )
  (rpn): RPNModule(
    (anchor_generator): AnchorGenerator(
      (cell_anchors): BufferList()
    )
    (head): RPNHead(
      (conv): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (cls_logits): Conv2d(1024, 9, kernel_size=(1, 1), stride=(1, 1))
      (bbox_pred): Conv2d(1024, 36, kernel_size=(1, 1), stride=(1, 1))
    )
    (box_selector_train): RPNPostProcessor()
    (box_selector_test): RPNPostProcessor()
  )
  (roi_heads): CombinedROIHeads(
    (box): ROIBoxHead(
      (feature_extractor): ResNet50Conv5ROIFeatureExtractor(
        (pooler): Pooler(
          (poolers): ModuleList(
            (0): ROIAlign(output_size=(14, 14), spatial_scale=0.0625, sampling_ratio=0)
          )
        )
        (head): ResNetHead(
          ... SOME RESNET LAYERS ...
        )
      )
      (predictor): FastRCNNPredictor(
        (avgpool): AvgPool2d(kernel_size=7, stride=7, padding=0)
        (cls_score): Linear(in_features=2048, out_features=21, bias=True)
        (bbox_pred): Linear(in_features=2048, out_features=84, bias=True)
      )
      (post_processor): PostProcessor()
    )
  )
)
```



Region Proposal Network



Input:

image;

targets - list(bbox)

bbox ~ (c, x, y, h, w)

Output:

predicted bboxes

losses

Region Proposal Network forward

```python
def forward(self, images, features, targets=None):

    objectness, rpn_box_regression = self.head(features)
    anchors = self.anchor_generator(images, features)

    if self.training:
        return self._forward_train(anchors, objectness, rpn_box_regression, targets)
    else:
        return self._forward_test(anchors, objectness, rpn_box_regression)

def _forward_train(self, anchors, objectness, rpn_box_regression, targets):
    if self.cfg.MODEL.RPN_ONLY:
        # When training an RPN-only model, the loss is determined by the
        # predicted objectness and rpn_box_regression values and there is
        # no need to transform the anchors into predicted boxes; this is an
        # optimization that avoids the unnecessary transformation.
        boxes = anchors
    else:
        # For end-to-end models, anchors must be transformed into boxes and
        # sampled into a training batch.
        with torch.no_grad():
            boxes = self.box_selector_train(
                anchors, objectness, rpn_box_regression, targets
            )
    loss_objectness, loss_rpn_box_reg = self.loss_evaluator(
        anchors, objectness, rpn_box_regression, targets
    )
    losses = {
        "loss_objectness": loss_objectness,
        "loss_rpn_box_reg": loss_rpn_box_reg,
    }
    return boxes, losses
```

Input:

image;

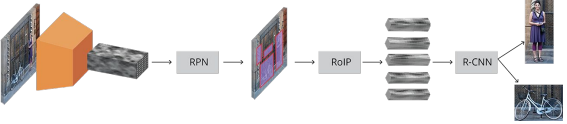
targets - list(bbox)

bbox ~ (c, x, y, h, w)

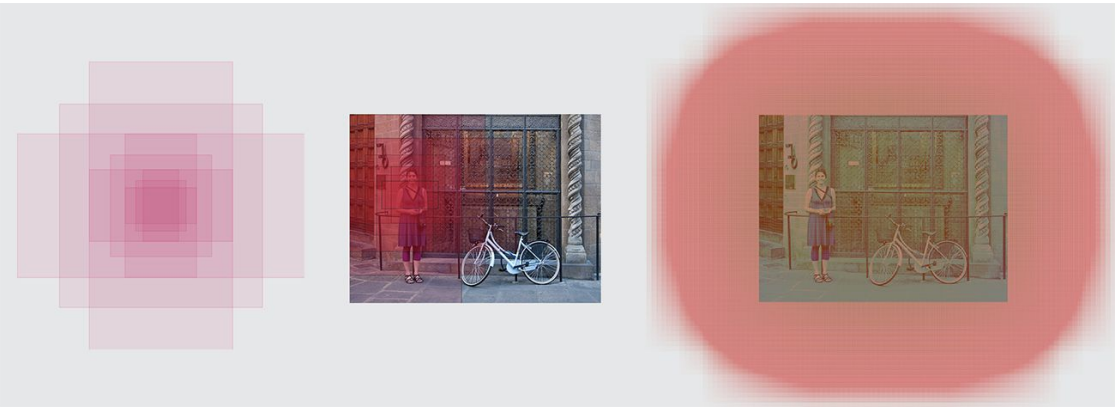
Output:

predicted bboxes

losses

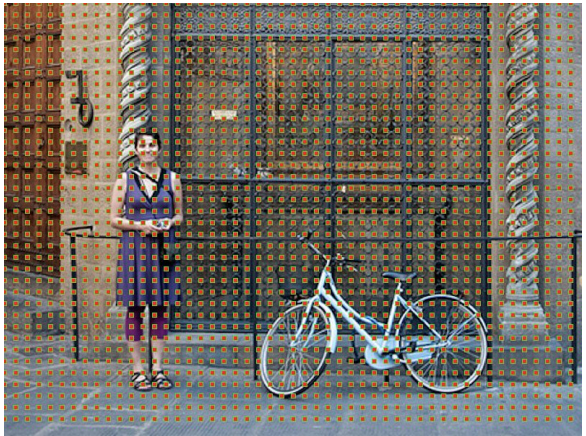


Region Proposal Network. Anchors



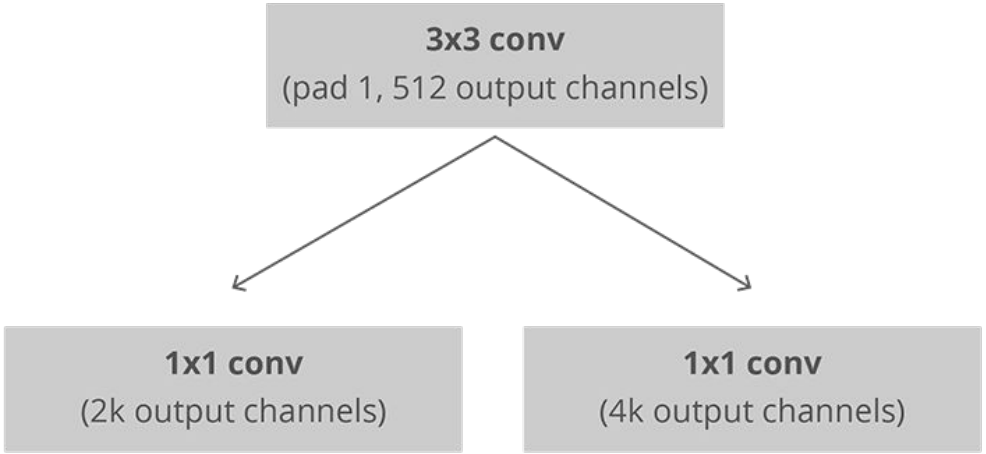
Get set of fixed-size boxes on image that corresponded to each point on feature map

```python
def forward(self, image_list, feature_maps):
    grid_sizes = [feature_map.shape[-2:] for feature_map in feature_maps]
    anchors_over_all_feature_maps = self.grid_anchors(grid_sizes)
    anchors = []
    for i, (image_height, image_width) in enumerate(image_list.image_sizes):
        anchors_in_image = []
        for anchors_per_feature_map in anchors_over_all_feature_maps:
            boxlist = BoxList(
                anchors_per_feature_map, (image_width, image_height), mode="xyxy"
            )
            self.add_visibility_to(boxlist)
            anchors_in_image.append(boxlist)
        anchors.append(anchors_in_image)
    return anchors
```



Region Proposal Network. Regressor and objectness

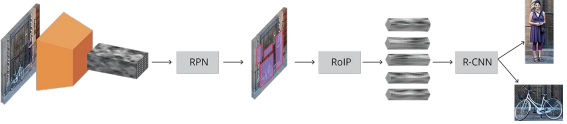
```
(rpn): RPNModule(
  (anchor_generator): AnchorGenerator(
    (cell_anchors): BufferList()
  )
  (head): RPNHead(
    (conv): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (cls_logits): Conv2d(1024, 9, kernel_size=(1, 1), stride=(1, 1))
    (bbox_pred): Conv2d(1024, 36, kernel_size=(1, 1), stride=(1, 1))
  )
  (box_selector_train): RPNPostProcessor()
  (box_selector_test): RPNPostProcessor()
)
```



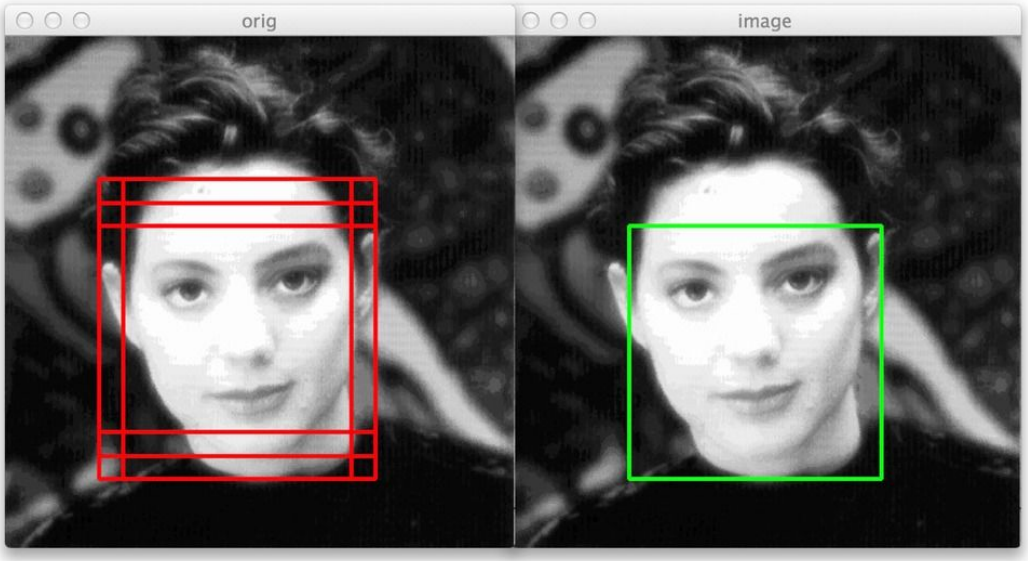
Regressor computes tilts for given anchors

$$t_x = (x - x_a)/w_a$$
$$t_y = (y - y_a)/h_a$$
$$t_w = \log(w/w_a)$$
$$t_w = \log(h/h_a)$$

Objectness predict probability of not-background



Non-maximum suppression



- Start from *bbox*(MAX) with highest IoU (intersection over union) score with its *ground-truth bbox* (predicted)
- exclude all bboxes with *iou*(*THIS*,*MAX*)⩾*THRESH*

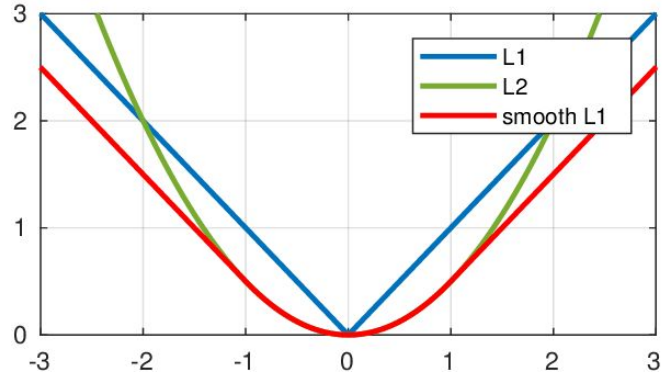
- Select the next prediction bbox with the highest IoU from the left bboxes until no bbox left.
- return top of the sorted bbox list:

list(Bbox)[:2000]

Region Proposal Network loss

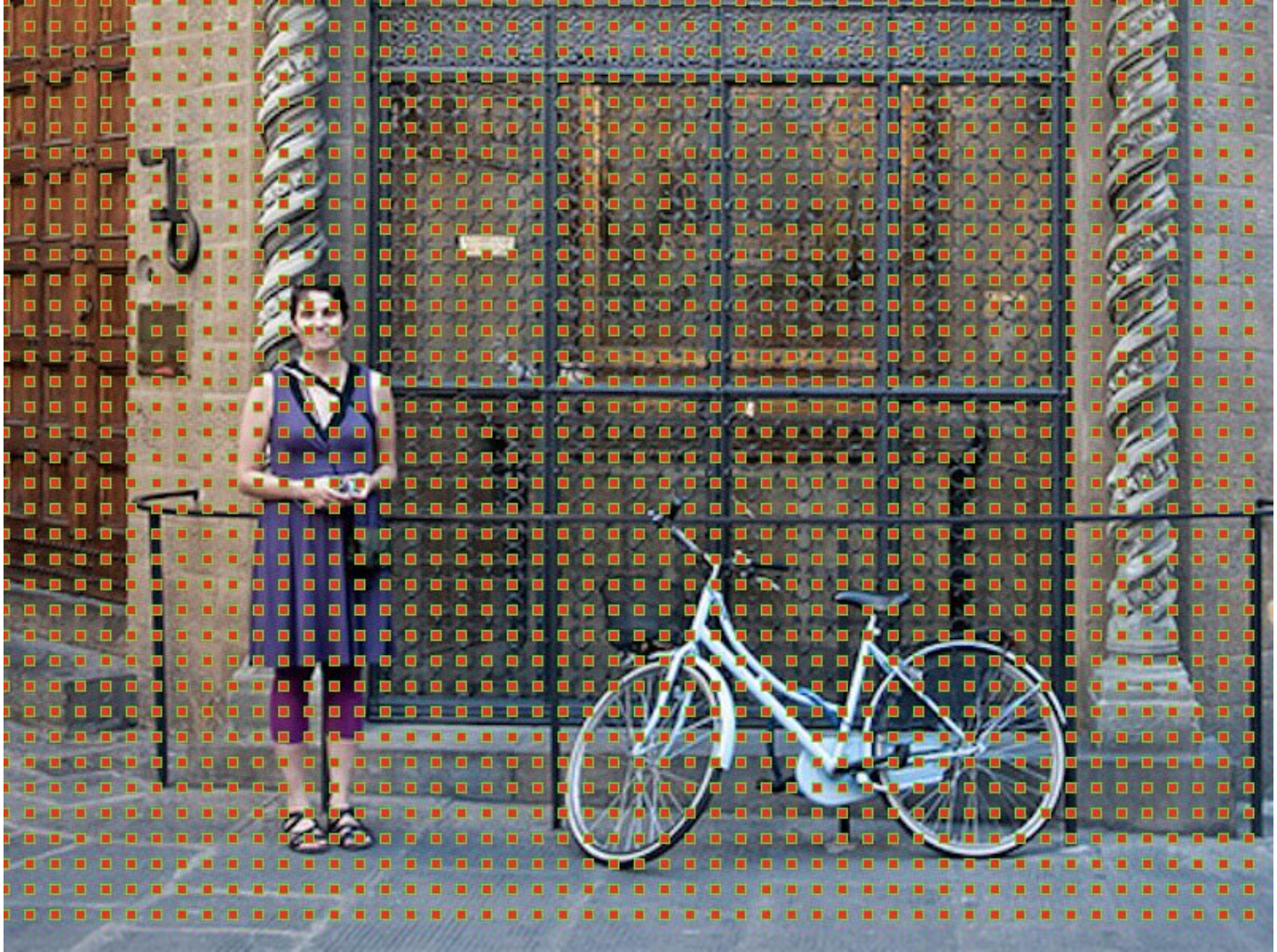
$$L(p, u, t, t^*) = L_{obj}(p, u) + [u > 0] \cdot L_{reg}(t, t^*)$$

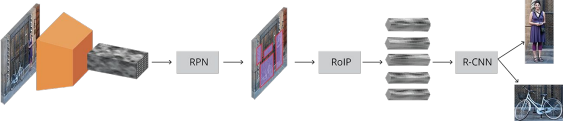
L_{obj} = cross-enthropy loss
L_{reg} = smooth L1 loss



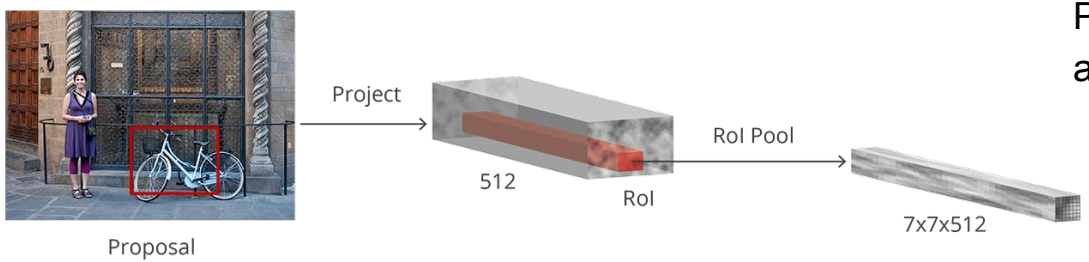
$$t_x = (x - x_a)/w_a$$
$$t_y = (y - y_a)/h_a$$
$$t_w = \log(w/w_a)$$
$$t_w = \log(h/h_a)$$

(x_a, y_a, w_a, h_a) – anchor bbox
t^* – tilts for ground-truth





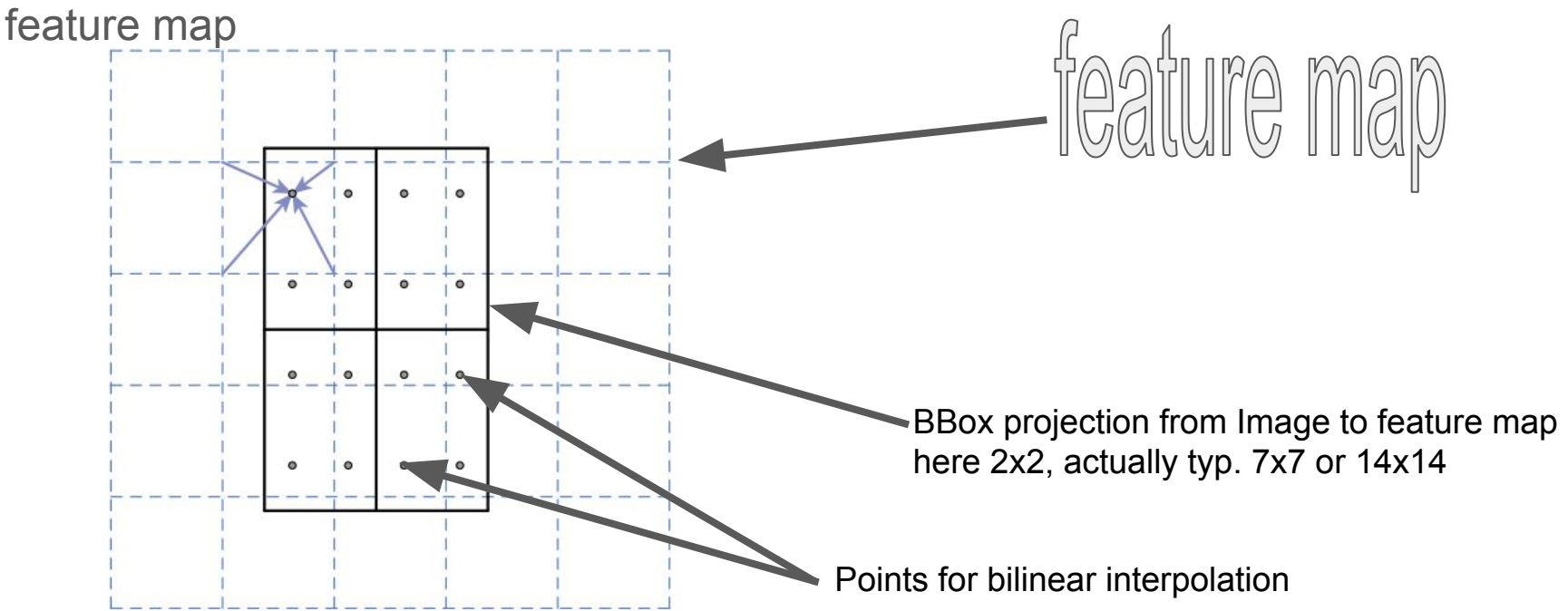
Regions of Interest(RoI) Pool/Allign

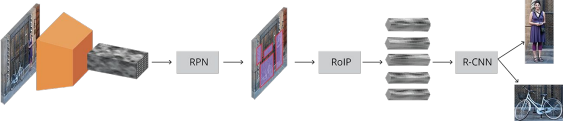


Project bounding box into feature space and then allign or pool to fixed size

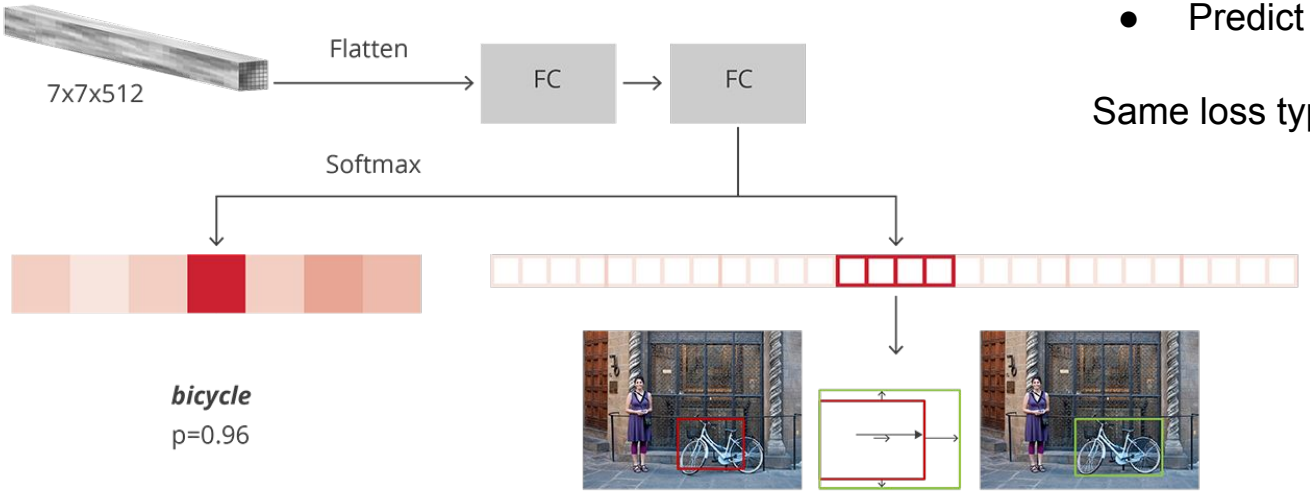
```
(roi_heads): CombinedROIHeads(
  (box): ROIBoxHead(
    (feature_extractor): ResNet50Conv5ROIFeatureExtractor(
      (pooler): Pooler(
        (poolers): ModuleList(
          (0): ROIAlign(output_size=(14, 14), spatial_scale=0.0625, sampling_ratio=0)
        )
      )
      (head): ResNetHead(
        ... SOME RESNET LAYERS ...
      )
    )
    (predictor): FastRCNNPredictor(
      (avgpool): AvgPool2d(kernel_size=7, stride=7, padding=0)
      (cls_score): Linear(in_features=2048, out_features=21, bias=True)
      (bbox_pred): Linear(in_features=2048, out_features=84, bias=True)
    )
    (post_processor): PostProcessor()
  )
)
```

Regions of Interest(RoI) Pool/Allign





(Regions of Interest) Head



- Finalise tilts for bounding box for each class
- Predict class probability

Same loss type as for RPN, but multiclass


person: 0.82
person: 0.89
person: 0.98
person: 0.83
person: 0.89
person: 0.80
person: 0.96
person: 0.99
person: 1.00
person: 0.99
cup: 0.99
cup: 1.00