به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین اول

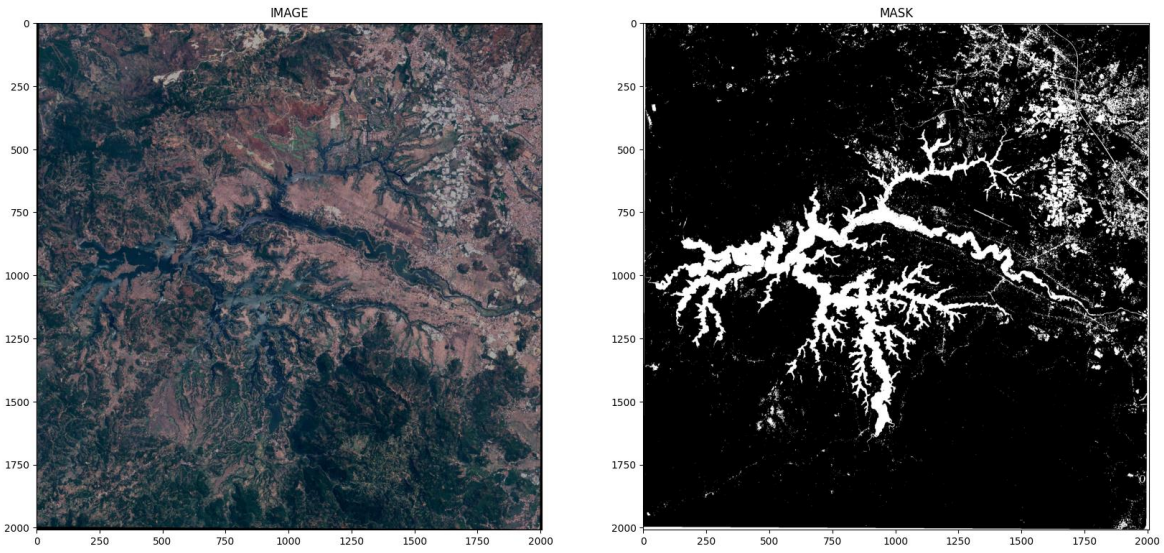
نام و نام خانوادگی	مهسا همت‌پناه – محمد هادی بابالو
شماره دانشجویی	۸۱۰۱۹۹۵۸۴ - ۸۱۰۱۹۹۳۸۰
تاریخ ارسال گزارش	۱۴۰۲.۰۹.۱۷

فهرست

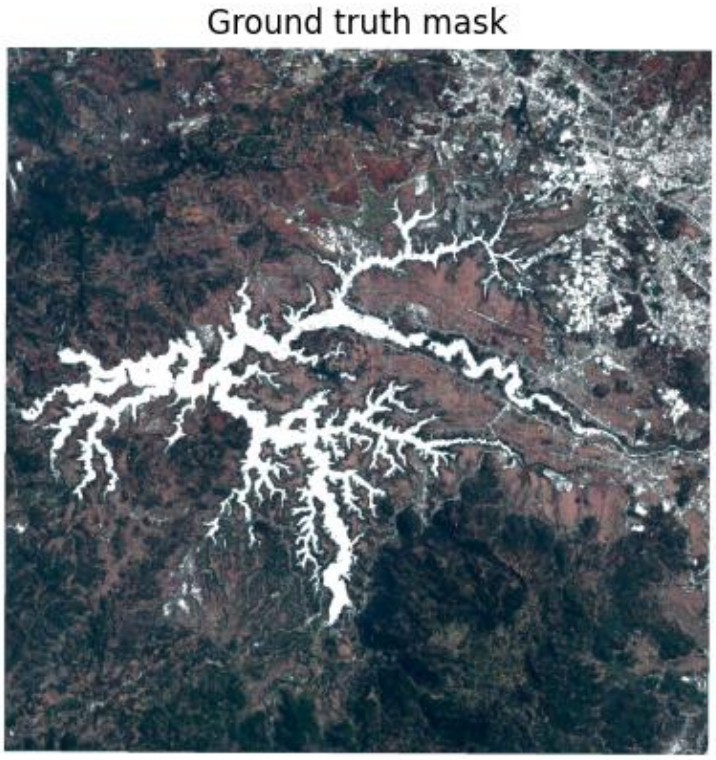
شکل‌ها

پاسخ ۱. SAM (Segment Anything Model)

۱-۱. آماده‌سازی مجموعه داده



شکل ۱-۱. نمونه‌ای از مجموعه داده، تصویر اصلی در کنار ماسک



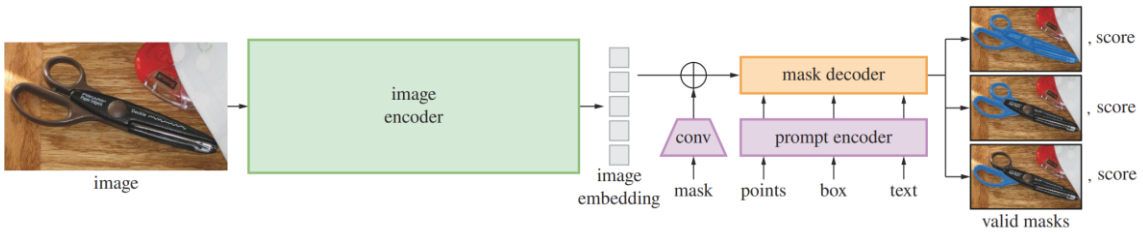
شکل ۱-۲. نمایش ماسک بر روی تصویر

Train-Test Split

```
test_ratio = 0.1
train_size = int((1 - test_ratio) * len(dataset))
test_size = len(dataset) - train_size
train_dataset, test_dataset = random_split(dataset, [train_size, test_size])
```

شکل ۱-۳. تقسیم داده‌ها به بخش آموزش و ارزیابی

۱-۲. بارگذاری مدل



شکل ۱-۴. معماری کلی شبکه SAM

به طور کلی هدف SAM این است که مدلی باشد که بتواند مدلی برای segmentation نه فقط کلاس‌های از پیش تعریف شده برای مدل، بلکه هر چیزی که در عکس وجود دارد باشد. این مدل عکسی را دریافت کرده و یک pixel-wise segmentation mask به عنوان خروجی تولید می‌کند به طوری که هر پیکسل در خروجی لیبلی اختصاص داده شده است که مشخص می‌کند این پیکسل متعلق به کدام object و یا حتی پس‌زمینه عکس است. این مدل متشکل از یک بخش image encoder، یک بخش prompt encoder و بخش mask generaator است. بخش image encoder مسئول استخراج ویژگی‌های سطح بالا از تصاویر ورودی با استفاده از backbone ای مثل شبکه ResNet و یا هر شبکه CNN شناخته شده دیگری است. بخش prompt encoder مسئول تبدیل prompt به بردار و استخراج اطلاعات متنی با استفاده از یک شبکه RNN و یا Transformer و تبدیل آن‌ها به یک نمایش عددی است. بخش mask generator هم که در واقع بخش اصلی SAM است، با گرفتن نتایج دو بخش قبلی و پیاده‌سازی یک mask decoder که به طور معمول یک convolutional decoder است، سعی در تولید ماسک خروجی می‌کند.

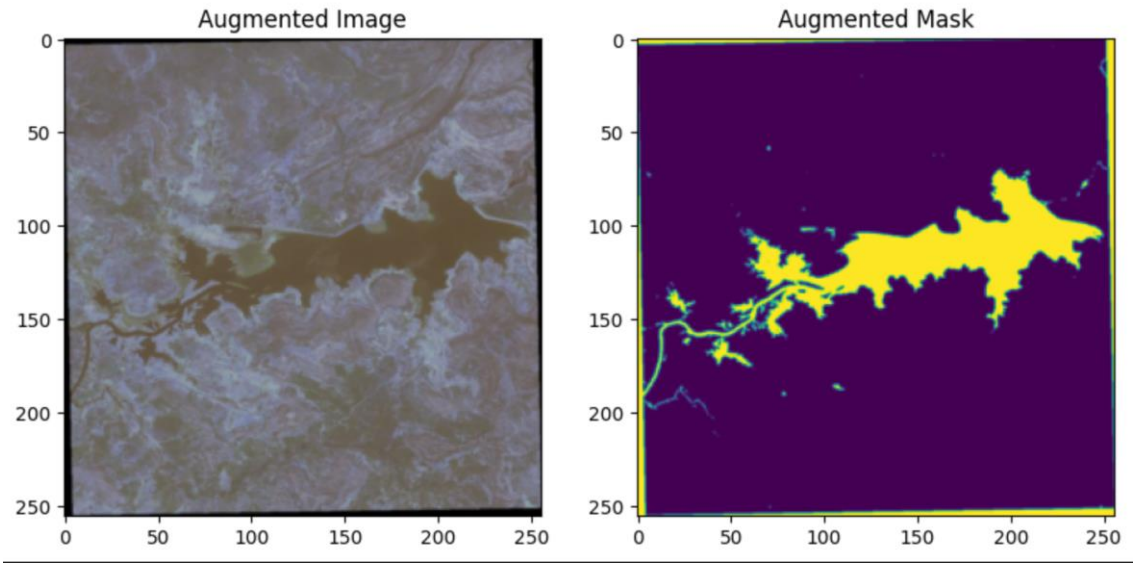
آموزش این شبکه معمولا از دو بخش Image Feature Learning و Prompt-Image Fusion and Mask Generation تشکیل شده است که در بخش اول در واقع backbone شبکه آموزش داده می‌شود و در بخش دوم هم بخش prompt encoder و mask decoder در کنار هم آموزش می‌بینند.

۱-۳. تقویت داده

```
Data Augmentation

augmentation_pipeline = A.Compose([
    #A.RandomCrop(width=256, height=256),
    A.HorizontalFlip(p=0.5),
    A.VerticalFlip(p=0.5),
    #A.Rotate(limit=30, p=0.5),
    A.RandomBrightnessContrast(p=0.2),
],is_check_shapes=False)

augmentated_dataset = ImagesDataset(new_images_list_aug, new_masks_list_aug, tfImg=tfImg, tfMsk=tfMsk, augmentation=augmentation_pipeline)
dataset = ConcatDataset([original_dataset, augmentated_dataset])
```

شکل ۱-۵. تقویت داده‌ها با استفاده از کتاب‌خانه Albumentations

می‌توان از روش‌های مختلف مثل flip کردن افقی یا عمودی عکس، تغییر روشنایی، تغییر کنتراست و یا چرخاندن عکس و غیره استفاده کرد که هر یک تاثیر خود را می‌گذارند. به طور مثال تغییر روشنایی عکس می‌تواند generalization مدل و عملکرد آن روی داده‌های دیده نشده را بالا ببرد، تغییر کنتراست می‌تواند به مدل برای تشخیص بهتر segment های مختلف عکس کمک کند. همچنین به طور مثال در این دیتاست خاص، ممکن است که مثلا در صورت مورب بودن یک رودخانه در عکسی bounding box متناظر آن دقیق نباشد و ناحیه بزرگی از عکس (بخش زیادی از پس‌زمینه به غیر از object مورد نظر ما) را پوشش دهد، در این صورت به طور مثال با چرخاندن عکس به مقدار کافی می‌توان این رودخانه را افقی‌تر (یا عمودی‌تر) کرده و در نتیجه bounding box بسیار دقیق‌تر و مناسب‌تری برای آن ارائه داد که این bounding box بسیار برای مدل ما کمک‌کننده‌تر باشد.



شکل ۱-۶. نمونه تصویر تقویت شده و ماسک متناظر آن

۱-۴. بهینه‌ساز، متریک و تابع هزینه

متریک IoU (Intersection over Union) نسبت مساحت اشتراک ماسک پیش‌بینی شده و ماسک واقعی به احتماع آن‌ها را اندازه می‌گیرد و عددی بین صفر و یک است. صفر به معنای همپوشانی نداشتن و یک به معنای انطباق کامل است.

```
def calculate_iou(true_mask, pred_mask):
    intersection = np.sum(pred_mask*true_mask)
    union = np.sum(pred_mask) + np.sum(true_mask) - intersection
    iou = np.mean(intersection/union)
    return round(iou, 4)
```

شکل ۱-۷. پیاده‌سازی معیار IoU

متریک Dice Coefficient میزان مشابهت ماسک پیشبینی‌شده و ماسک واقعی را با محاسبه دو برابر نسبت مساحت همپوشانی ماسک‌ها به مجموع مساحت ماسک‌ها اندازه‌گیری می‌کند.

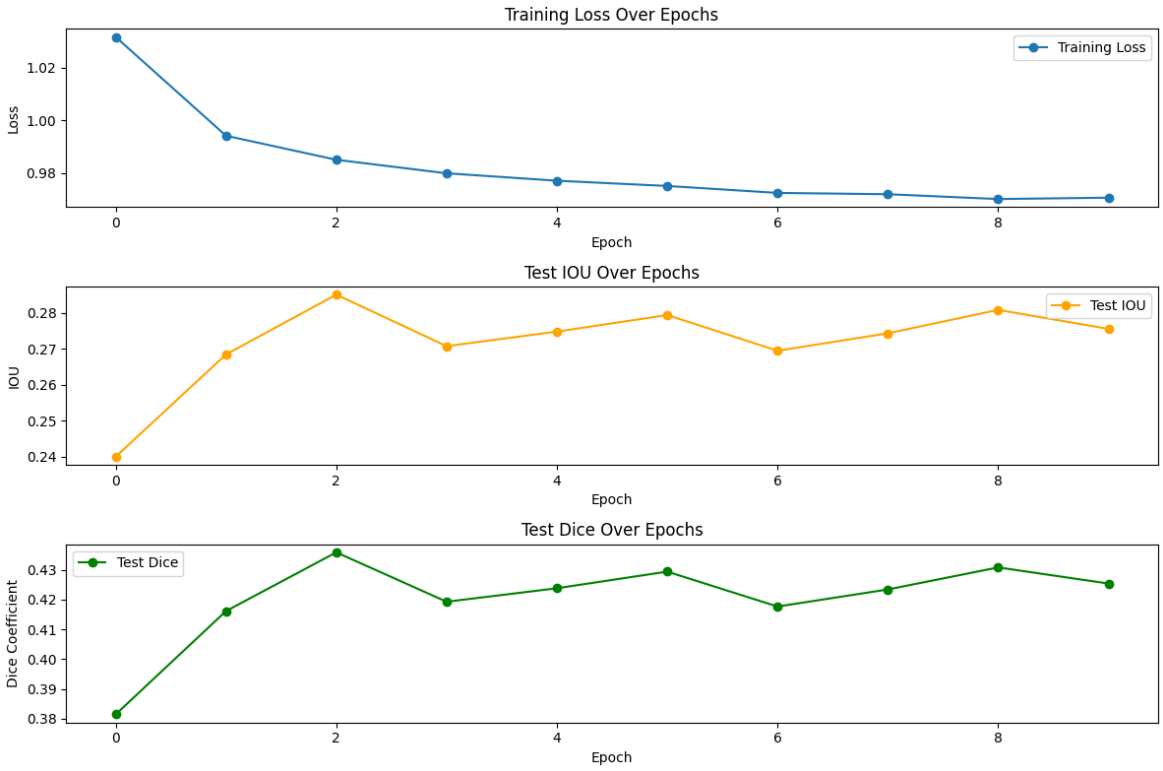
```
def calculate_dice_coefficient(true_mask, pred_mask):
    intersect = np.sum(pred_mask*true_mask)
    total_sum = np.sum(pred_mask) + np.sum(true_mask)
    dice = np.mean(2*intersect/total_sum)
    return round(dice, 4)
```

شکل ۱-۸. پیاده‌سازی معیار Dice

در این پروژه از بهینه‌ساز Adam به دلیل مزایای مختلف آن مثل Adaptive learning rate و سرعت خوب همگرایی آن استفاده کرده‌ایم. همچنین برای loss function انتخاب‌های متعددی مثل Pixel-wise Cross Entropy Loss، Consistency Loss، Boundary Loss، Jaccard/Intersection over Union (IoU) و DiceCELoss وجود دارد که هر یک از آن‌ها مزایا و معایب خود را دارند و در جای خود استفاده می‌شوند. در این پروژه ما از DiceCELoss که ترکیبی هایبرید از Dice Loss و Cross Entropy است به دلیل دقت نهایی بهتر مدل و همچنین generalization خوب آن طبق بررسی‌های به عمل آمده توسط پژوهشگران دیگر استفاده کرده‌ایم.

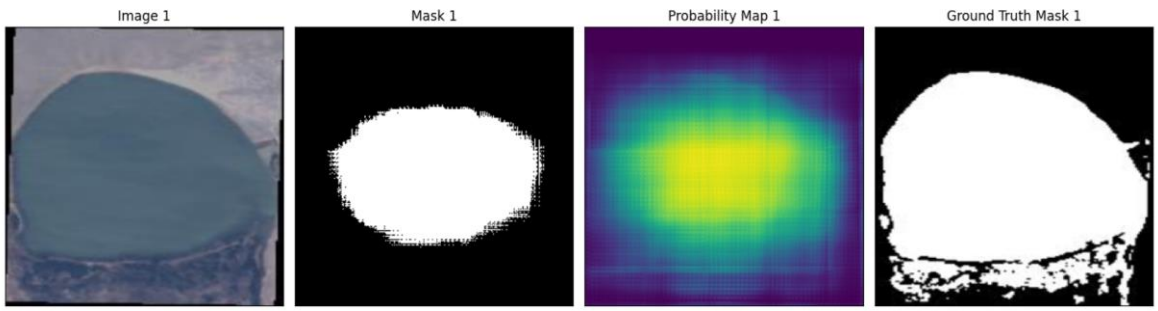
۱-۵. Fine-Tune کردن مدل

۱-۶. ارزیابی نتایج



شکل ۱-۹. متریک‌ها و loss در حین آموزش مدل

همانطور که دیده می‌شود loss ما در حین فرایند آموزش و fine-tune کردن مدل در حال کاهش است که این رفتاری است که انتظار داشتیم. همچنین هر دو متریک IoU و Dice بر روی داده‌های ارزیابی ما در حین آموزش در حال بهبود هستند که نشان‌دهنده این است که ماسک‌های پیشبینی‌شده به سمت همپوشانی بیشتر با ماسک‌های واقعی می‌روند. همچنین نمونه‌ای از خروجی مدل را در شکل ۱-۱۰ می‌توانید مشاهده کنید.



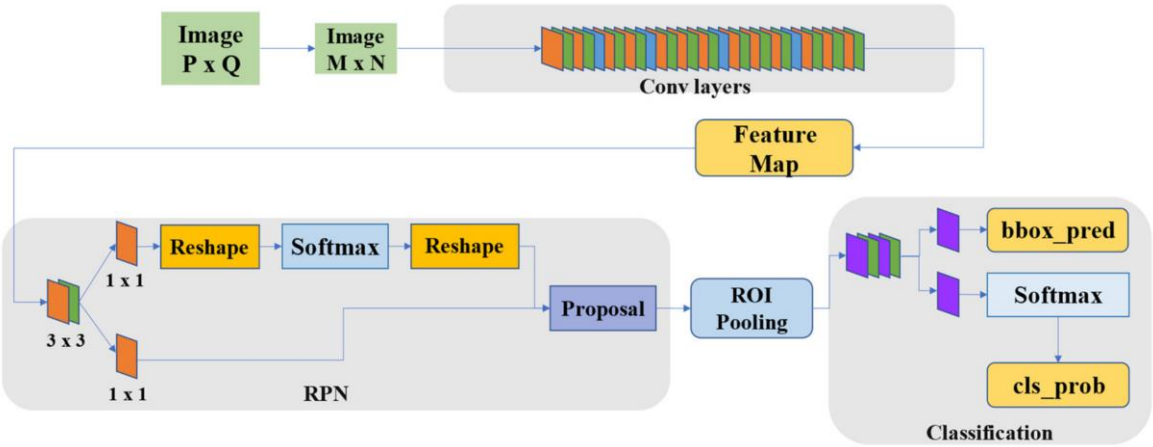
شکل ۱-۱۰. نمونه ورودی، ماسک پیشبینی‌شده، توزیع احتمال و ماسک واقعی

پاسخ ۲ – آشنایی و پیاده‌سازی مدل Faster R-CNN

۲-۱. توضیحات مدل‌ها

بهبودهای Faster RCNN نسبت به دو مدل دیگر:

- بخش RPN در Faster RCNN به صورت شبکه‌ای که جزئی از شبکه کلی ما است که پیشنهادها را تولید می‌کند پیاده‌سازی شده ولی به طور مثال در Fast RCNN، این proposal ها با استفاده از الگوریتمی مثل Selective Search تولید شده و مستقیما به شبکه داده می‌شوند. به همین دلیل این بخش در Faster RCNN هم سریع‌تر و هم از لحاظ محاسباتی سبک‌تر است.
- در RCNN و Fast RCNN بخش آموزش به صورت جدا جدا باید ابتدا برای region proposal انجام شود و سپس شبکه object detection با استفاده از نتیجه به دست آمده از آموزش قبلی، آموزش داده می‌شود. در حالی که در Faster RCNN به دلیل یکپارچه بودن فرایند آموزش، مدل را بهینه‌تر می‌کند.
- لایه RoI Pooling در Faster RCNN نسبت به Fast RCNN بهبود داشته است.
- همچنین از لحاظ بهینگی، سرعت و همچنین دقت نهایی مدل، Faster RCNN نسبت به RCNN و Fast RCNN عملکرد بهتری دارد.



شکل ۲-۱. معماری کلی Faster R-CNN

Conv Layer

این بخش شبکه که به عنوان backbone آن هم شناخته می‌شود، مسئول گرفتن عکس‌های ورودی و تبدیل و استخراج ویژگی (feature extraction) از آن‌ها و تولید feature map ورودی داده شده به

RPN است. در این بخش معمولا از معماری یک شبکه CNN شناخته شده مثل VGG یا ResNet که روی یک دیتاست شناخته شده مثل AlexNet یا Pascal، pretrain شده است استفاده می‌شود.

Region Proposal Network

این بخش به عنوان یکی از مهم‌ترین بخش‌های این شبکه و از بخش‌های متمایزکننده آن محسوب می‌شود. هدف اصلی این بخش تولید مجموعه‌ای از پیشنهادهای bounding box کاندیدا بر روی تصویر ورودی است. این نواحی در واقع نواحی‌ای هستند که پتانسیل این را دارند که object ای در آن‌ها وجود داشته باشد و برای ادامه پردازش به بخش بعدی شبکه داده می‌شوند. در این بخش از شبکه از sliding window ها، anchor ها، دو شبکه مجزا برای classification و regression و Non-Maximum Suppression استفاده شده و در نهایت ساخت proposal ها انجام می‌شود.

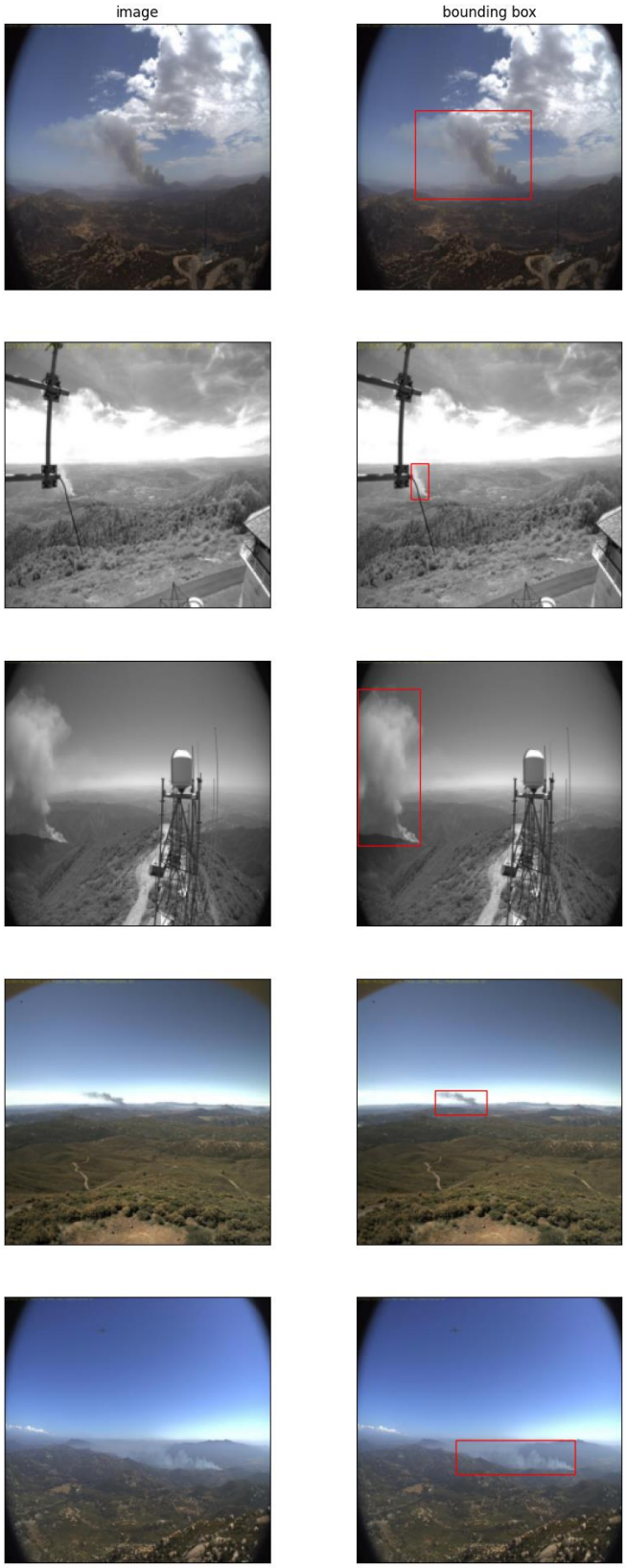
RoI Pooling

کارکرد و هدف اصلی بخش RoI (Region of Intrest) Pooling استخراج و تبدیل proposal های خروجی داده شده از شبکه RPN، که به دلیل بخش آخر RPN که proposal ها را تولید می‌کند دارای ابعاد متغیر هستند، به ابعاد ثابت است تا بتوان آن‌ها را به بخش آخر شبکه که یک Fully Connected Network است ورودی داد و مشکلی برای شبکه در مرحله آخر معماری آن ایجاد نشود.

Classification

بخش آخر شبکه Faster RCNN، مسئول تولید پیش‌بینی‌های نهایی مدل و refine کردن bounding box های به دست آمده از مراحل قبل است. این مرحله خروجی یا سایز ثابت را از RoI دریافت کرده و آن‌ها را به خروجی‌های نهایی مدل که شامل bounding box های نهایی و نتیجه classification هر کدام از آن‌ها است تبدیل کرده و خروجی‌های نهایی ما از مدل را تولید می‌کند.

۲-۲. پیش‌پردازش



شکل ۲-۲. نمونه تصاویر به همراه bounding box

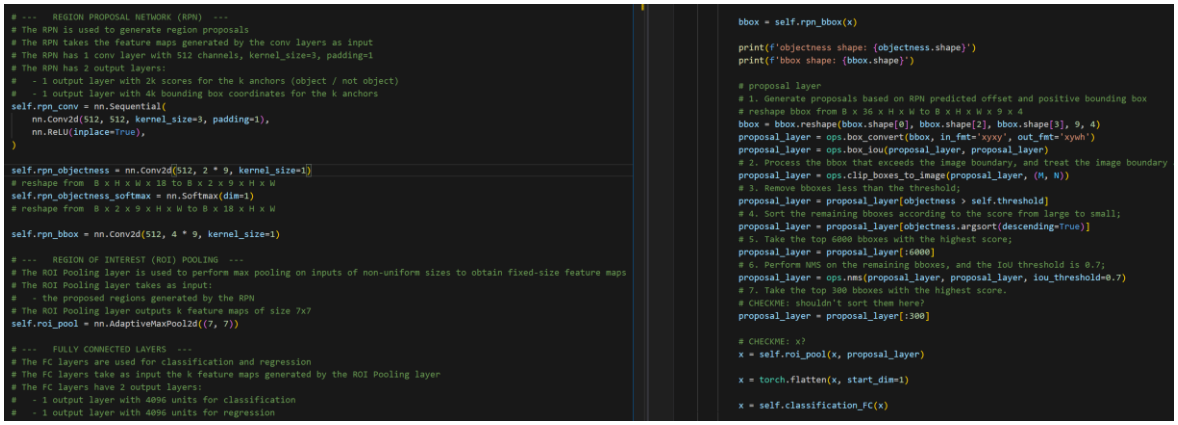
```
if self.transform:
    image = self.transform(image)
    # should change annotation as well in case of resizing
    annotation['boxes'][0] = annotation['boxes'][0] * M / annotation['width']
    annotation['boxes'][1] = annotation['boxes'][1] * N / annotation['height']
    annotation['boxes'][2] = annotation['boxes'][2] * M / annotation['width']
    annotation['boxes'][3] = annotation['boxes'][3] * N / annotation['height']
    annotation['width'] = M
    annotation['height'] = N
```

شکل ۲-۳. تغییر سایز تصاویر و برچسب‌های متناظر آن‌ها

۲-۳. آموزش شبکه

```
# ---   THE CONV LAYERS  ---
# The Conv layers contains 13 conv layers, 13 ReLU layers and 4 pooling layers
# All the conv layers are: kernel_size=3, pad=1, stride=1
# All the pooling layers are: kernel_size=2, stride=2
self.features = nn.Sequential(
    nn.Conv2d(3, 64, kernel_size=3, padding=1, stride=1),
    nn.ReLU(inplace=True),

    nn.Conv2d(64, 64, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),

    nn.MaxPool2d(kernel_size=2, stride=2),

    nn.Conv2d(64, 128, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),

    nn.Conv2d(128, 128, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),

    nn.MaxPool2d(kernel_size=2, stride=2),

    nn.Conv2d(128, 256, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),

    nn.Conv2d(256, 256, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),

    nn.Conv2d(256, 256, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),

    nn.MaxPool2d(kernel_size=2, stride=2),

    nn.Conv2d(256, 512, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),

    nn.Conv2d(512, 512, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),

    nn.Conv2d(512, 512, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),

    nn.MaxPool2d(kernel_size=2, stride=2),

    nn.Conv2d(512, 512, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),

    nn.Conv2d(512, 512, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),

    nn.Conv2d(512, 512, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),
)
```

شکل ۲-۴. بخش Conv شبکه



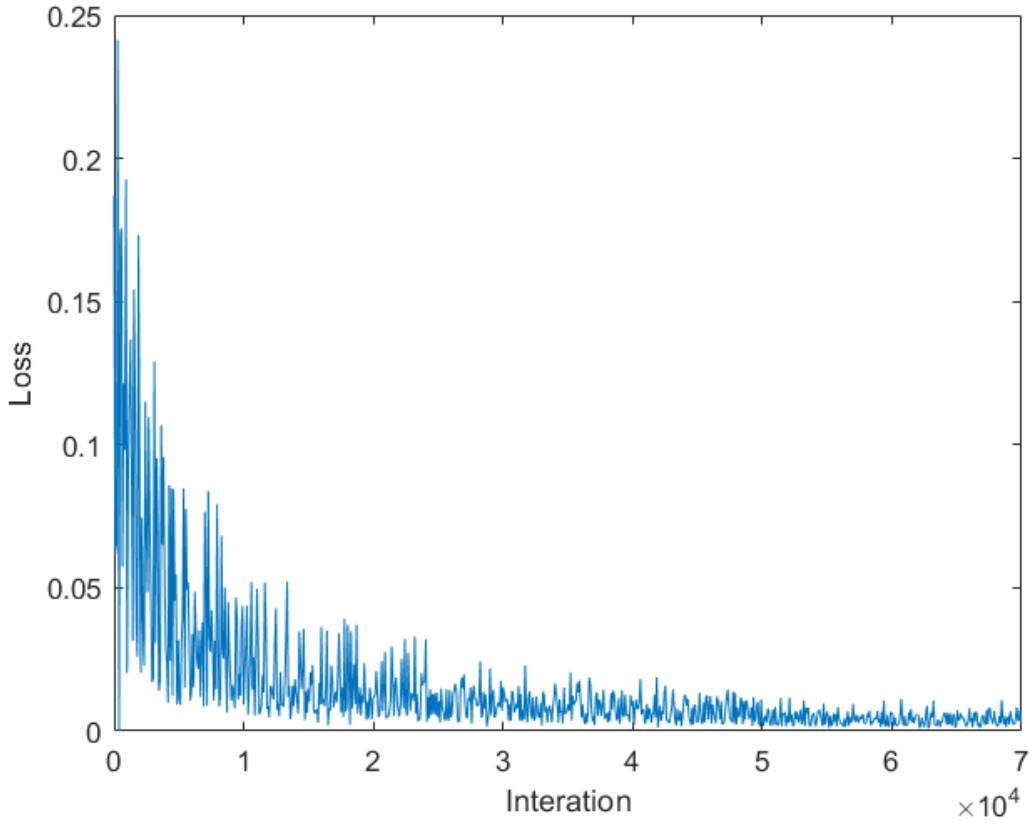
```
# ---   REGION PROPOSAL NETWORK (RPN)  ---
# The RPN is used to generate region proposals
# The RPN takes the feature maps generated by the conv layers as input
# The RPN has 1 conv layer with 512 channels, kernel_size=3, padding=1
# The RPN has 2 output layers:
#   - 1 output layer with 2k scores for the k anchors (object / not object)
#   - 1 output layer with 4k bounding box coordinates for the k anchors
self.rpn_conv = nn.Sequential(
    nn.Conv2d(512, 512, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),
)

self.rpn_objectness = nn.Conv2d(512, 2 * 9, kernel_size=1)
# reshape from  B x H x W x 18 to B x 2 x 9 x H x W
self.rpn_objectness_softmax = nn.Softmax(dim=1)
# reshape from  B x 2 x 9 x H x W to B x 18 x H x W

self.rpn_bbox = nn.Conv2d(512, 4 * 9, kernel_size=1)

# ---   REGION OF INTEREST (ROI) POOLING  ---
# The ROI Pooling layer is used to perform max pooling on inputs of non-uniform sizes to obtain fixed-size feature maps
# The ROI Pooling layer takes as input:
#   - the proposed regions generated by the RPN
# The ROI Pooling layer outputs k feature maps of size 7x7
self.roi_pool = nn.AdaptiveMaxPool2d((7, 7))

# ---   FULLY CONNECTED LAYERS  ---
# The FC layers are used for classification and regression
# The FC layers take as input the k feature maps generated by the ROI Pooling layer
# The FC layers have 2 output layers:
#   - 1 output layer with 4096 units for classification
#   - 1 output layer with 4096 units for regression
```

```
bbox = self.rpn_bbox(x)

print(f'objectness shape: {objectness.shape}')
print(f'bbox shape: {bbox.shape}')

# proposal layer
# 1. Generate proposals based on RPN predicted offset and positive bounding box
# reshape bbox from B x 36 x H x W to B x H x W x 9 x 4
bbox = bbox.reshape(bbox.shape[0], bbox.shape[2], bbox.shape[3], 9, 4)
proposal_layer = ops.box_convert(bbox, in_fmt='xyxy', out_fmt='xywh')
proposal_layer = ops.box_iou(proposal_layer, proposal_layer)
# 2. Process the bbox that exceeds the image boundary, and treat the image boundary
proposal_layer = ops.clip_boxes_to_image(proposal_layer, (M, N))
# 3. Remove bboxes less than the threshold;
proposal_layer = proposal_layer[objectness > self.threshold]
# 4. Sort the remaining bboxes according to the score from large to small;
proposal_layer = proposal_layer[objectness.argsort(descending=True)]
# 5. Take the top 6000 bboxes with the highest score;
proposal_layer = proposal_layer[:6000]
# 6. Perform NMS on the remaining bboxes, and the IoU threshold is 0.7;
proposal_layer = ops.nms(proposal_layer, proposal_layer, iou_threshold=0.7)
# 7. Take the top 300 bboxes with the highest score.
# CHECKME: shouldn't sort them here?
proposal_layer = proposal_layer[:300]

# CHECKME: x?
x = self.roi_pool(x, proposal_layer)

x = torch.flatten(x, start_dim=1)

x = self.classification_FC(x)
```

شکل ۲-۵. بخش RPN و RoI Pooling شبکه

```
# ---   FULLY CONNECTED LAYERS  ---
# The FC layers are used for classification and regression
# The FC layers take as input the k feature maps generated by the ROI Pooling layer
# The FC layers have 2 output layers:
#   - 1 output layer with 4096 units for classification
#   - 1 output layer with 4096 units for regression

self.classification_FC = nn.Sequential(
    # nn.Dropout(),

    nn.Linear(512 * 7 * 7, 4096),
    nn.ReLU(inplace=True),

    # nn.Dropout(),

    nn.Linear(4096, 4096),
    nn.ReLU(inplace=True),
)

self.classification_classifier = nn.Sequential(
    nn.Linear(4096, self.num_classes),
    nn.Softmax(dim=1),
)

self.classification_bbox_regressor = nn.Sequential(
    nn.Linear(4096, self.num_classes * 4),
)
```

شکل ۲-۶. بخش Fully Connected شبکه



شکل ۲-۷. نمودار loss در حین آموزش شبکه Faster RCNN

۲-۴. بررسی داده‌های تست

همانطور که در نتایج مقاله هم ذکر شده است و با توجه به استفاده شدن از معماری Faster RCNN در تسک‌های مختلف image segmentation، همانطور که انتظار داریم این مدل عملکرد خوب و قابل قبولی از خود به جا می‌گذارد. البته که عملکرد و دقت مدل به فاکتورهای متفاوتی مثل اندازه و کیفیت دیتاست، پیش‌پردازش‌های انجام شده بر روی آن، backbone استفاده شده در مدل، هایپرپارامترهای مدل مثل تعداد anchor ها یا aspect ratio ها و ... بستگی خواهد داشت.

از ضعف‌های مدل Faster RCNN می‌توان به سرعت آن اشاره کرد. درست است که این مدل نسبت به RCNN و Fast RCNN سرعت بسیار بیشتری دارد اما همچنان کل این شبکه مخصوصا در بخش inference نسبتا کند عمل می‌کند. برای بهبود این بخش می‌توان از معماری‌های جدیدتر و پیشرفته‌تری مثل YOLO (You Only Look Once) استفاده کرد. همچنین بخش RPN به تنظیمات anchor box ها مثل aspect ratio ها حساس است و ممکن است برای اشیای خارج از آن‌ها خوب عمل نکند، برای حل این مشکل هم می‌توان از شبکه‌های دیگر مثل Feature Pyramid Networks (FPN) و یا Region-based Fully Convolutional Networks (R-FCN) استفاده کرد. این مدل همچنین در

تشخیص اشیای کوچک و به خصوص نزدیک به هم نیز ممکن است با مشکل مواجه شود که یکی از کارها برای بهبود آن می‌تواند استفاده از RoIAllign باشد.