**Detection**

1. Các thuật toán
   1. RCNN

* Sử dụng Selective search để đưa ra các region proposals để dự đoán các vùng có khả năng chứa vật thể ở trong.
* Tạo fearture vector từ các region proposals bằng cách sử dụng các mạng CNN như Alex-Net hoặc VGG16
* Dùng SVM để phân loại các feature vector.

Nhược: Với mỗi ảnh ta cần phân loại class cho 2000 region proposals => thời gian train lâu => không thể áp dụng cho real time

* 1. Fast R-CNN

Vẫn dùng selective search để lấy ra các region proposal nhưng cho nguyên bức ảnh vào một mạng ConvNet để tao ra convolution feature map. Lấy các feature tương ứng với region proposal flatten và thêm 2 lớp Fully connected để dự đoán lớp của region.

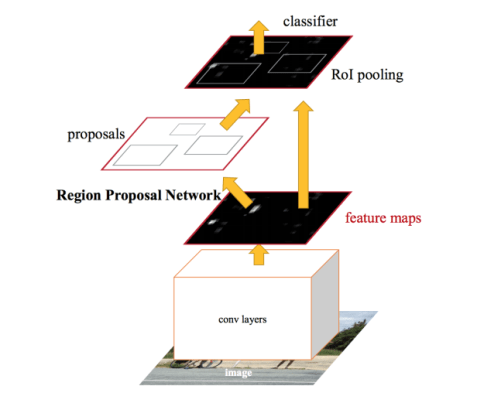
Nhưng do các region có kích thước khác nhau nên sẽ tạo ra các feature vector có kích thước khác nhau

* Sử dụng Region of Interest (ROI) pooling để đưa ra output có kích thước cố định. (Chia các region thành các vung, ở mỗi vùng lấy max pooling)

So sánh với RCNN thì thời gian tính region proposals rất lâu và làm chậm thuật toán => phải dùng phương pháp khác để thay thế selective search.

* 1. Faster R-CNN

Thêm một mạng CNN mới gọi là Region Proposal Network (RPN) để tìm các region proposal. Đưa cả hình ảnh vào để lấy feature map, dùng Region Proposal Network (RPN) để xác định được vị trí của các region proposals. Sử dụng Anchor để biểu diễn các region proposals do các giá trị x,y của bounding box được dự đoán có khả năng ra ngoài bức ảnh. Kiến trúc mạng Faster R-CNN được thể hiện ở hình sau.



* 1. Mask R-CNN

Sử dụng ResNet101 để rút trích đặc trưng từ hình ảnh và dung Region Proposal Network để tạo Region of interest (ROI). *Mask R-CNN tạo segmentation mask.*

1. Cài đặt Faster RCNN

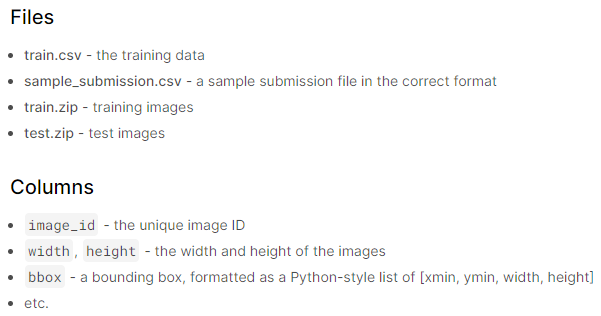
## 2.1 Dataset

Dữ liệu được lấy từ cuộc thi global wheat detection của Kaggle.

Link cuộc thi: <https://www.kaggle.com/c/global-wheat-detection/data>

Mục tiêu của cuộc thi là dự đoán bounding box xung quanh phần hạt lúa mì. Bộ dữ liệu gồm 3422 ảnh train và 10 ảnh test. Trong 3422 ảnh có tổng cộng 147793 hạt lúa mì.

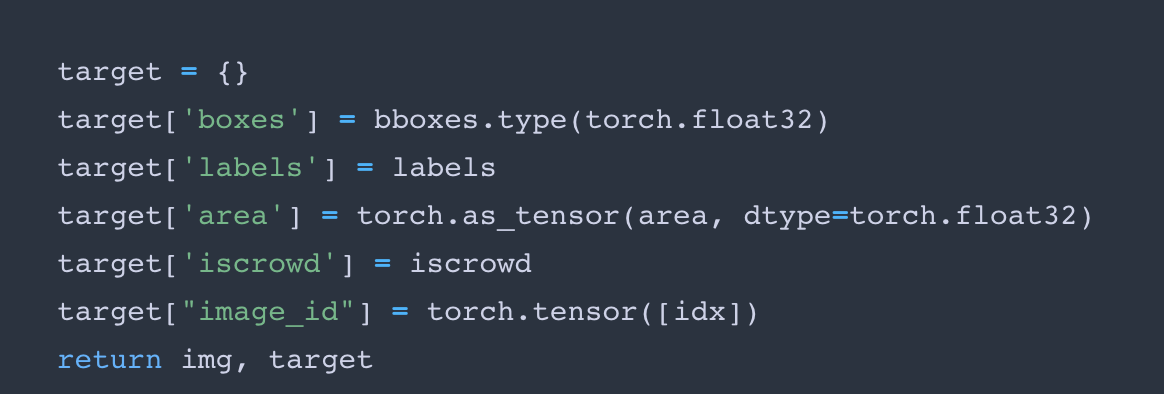
Việc cài đặt mô hình được sử dụng dữ liệu của tập train và chia dữ liệu của tập train thành 3 phần: train, validation và test.



## 2.2 Cách cài đặt (sử dụng pytorch)

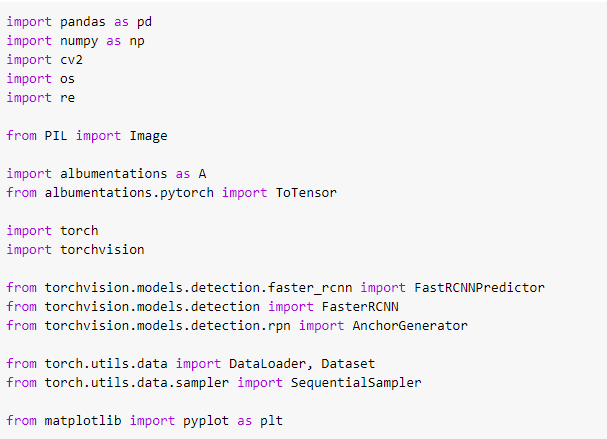
### 2.2.1 Chuẩn bị dữ liệu

Model Faster R-CNN nhận cặp (x\_train, y\_train) theo định dạng (image,target) với target gồm các trường thông tin như sau:



* Chuyển dữ liệu về dạng này để có thể train mô hình.

Import các thư viện cần thiết:



Sử dụng pandas để đọc file train.csv chứa thông tin bbox của tập train

train\_df = pd.read\_csv('train.csv')

Xây dựng lớp WheatDataset để chuyển dữ liệu theo đúng đinh dạng của mô hình.

class WheatDataset(Dataset):

    def \_\_init\_\_(self, dataframe, image\_dir, transforms=None):

        super().\_\_init\_\_()

        self.image\_ids = dataframe['image\_id'].unique()

        self.df = dataframe

        self.image\_dir = image\_dir

        self.transforms = transforms

    def \_\_getitem\_\_(self, index: int):

        # take id of image

        image\_id = self.image\_ids[index]

        records = self.df[self.df['image\_id'] == image\_id]

        # print ('records: ',records)

        # read image

        image = cv2.imread(f'{self.image\_dir}/{image\_id}.jpg', cv2.IMREAD\_COLOR)

        image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB).astype(np.float32)

        image /= 255.0

        boxes = records[['x', 'y', 'w', 'h']].values

        boxes[:, 2] = boxes[:, 0] + boxes[:, 2]

        boxes[:, 3] = boxes[:, 1] + boxes[:, 3]

        # print('boxes: ',boxes)

        area = (boxes[:, 3] - boxes[:, 1]) \* (boxes[:, 2] - boxes[:, 0])

        area = torch.as\_tensor(area, dtype=torch.float32)

        # there is only one class

        labels = torch.ones((records.shape[0],), dtype=torch.int64)

        # suppose all instances are not crowd

        iscrowd = torch.zeros((records.shape[0],), dtype=torch.int64)

        target = {}

        target['boxes'] = boxes

        target['labels'] = labels

        target['image\_id'] = torch.tensor([index])

        target['area'] = area

        target['iscrowd'] = iscrowd

        if self.transforms:

            sample = {

                'image': image,

                'bboxes': target['boxes'],

                'labels': labels

            }

            sample = self.transforms(\*\*sample)

            image = sample['image']

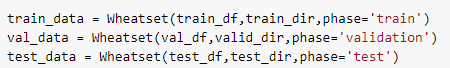
            target['boxes'] = torch.stack(tuple(map(torch.tensor, zip(\*sample['bboxes'])))).permute(1, 0)

        return image, target, image\_id

    def \_\_len\_\_(self) -> int:

        return self.image\_ids.shape[0]

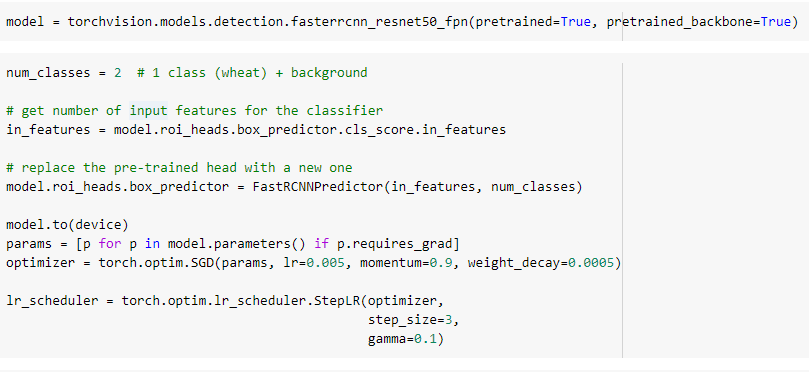
Khởi tạo Dataset (đưa model về dạng phù hợp với mô hình) và Dataloader (chia dataset vừa tạo thành các phần nhỏ) để chuẩn bị cho việc train mô hình.





### 2.2.2 Train model

Load pretrain model của Faster R-CNN. Sử dụng optimizer là SGD với learning rate là 0.005 và momentum là 0.9.



Hàm train model:

def train\_one\_epoch(model, optimizer, data\_loader, device, epoch, print\_freq):

    model.train()

    metric\_logger = utils.MetricLogger(delimiter="  ")

    metric\_logger.add\_meter('lr', utils.SmoothedValue(window\_size=1, fmt='{value:.6f}'))

    header = 'Epoch: [{}]'.format(epoch)

    lr\_scheduler = None

    if epoch == 0:

        warmup\_factor = 1. / 1000

        warmup\_iters = min(1000, len(data\_loader) - 1)

        lr\_scheduler = utils.warmup\_lr\_scheduler(optimizer, warmup\_iters, warmup\_factor)

    for images, targets in metric\_logger.log\_every(data\_loader, print\_freq, header):

        images = list(image.to(device) for image in images)

        targets = [{k: v.to(device) for k, v in t.items()} for t in targets]

        loss\_dict = model(images, targets)

        losses = sum(loss for loss in loss\_dict.values())

        train\_loss.append(losses.item())

        # reduce losses over all GPUs for logging purposes

        loss\_dict\_reduced = utils.reduce\_dict(loss\_dict)

        losses\_reduced = sum(loss for loss in loss\_dict\_reduced.values())

        loss\_value = losses\_reduced.item()

        if not math.isfinite(loss\_value):

            print("Loss is {}, stopping training".format(loss\_value))

            print(loss\_dict\_reduced)

            sys.exit(1)

        optimizer.zero\_grad()

        losses.backward()

        optimizer.step()

        if lr\_scheduler is not None:

            lr\_scheduler.step()

        metric\_logger.update(loss=losses\_reduced, \*\*loss\_dict\_reduced)

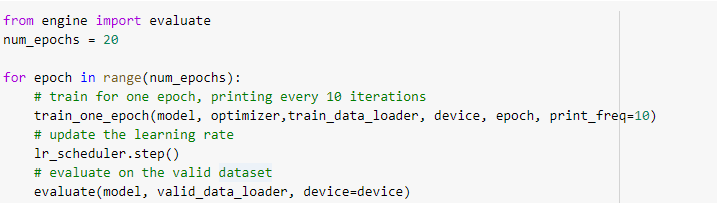
        metric\_logger.update(lr=optimizer.param\_groups[0]["lr"])

    epoch\_train\_loss = np.mean(train\_loss)

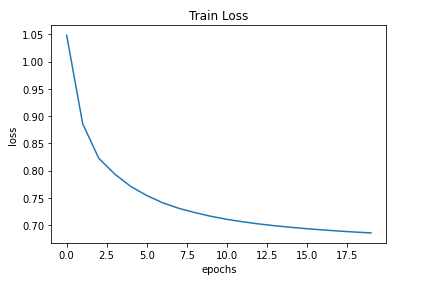
    total\_train\_loss.append(epoch\_train\_loss)

    return metric\_logger

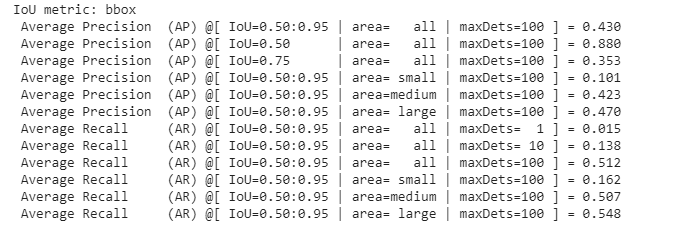
Train model với 20 epoch



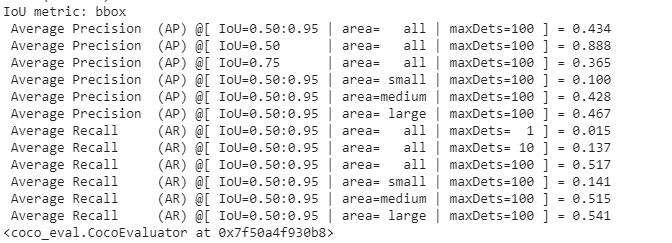
Kết quả sau khi train model:

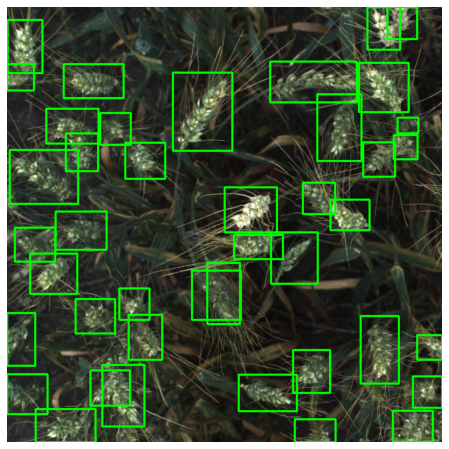
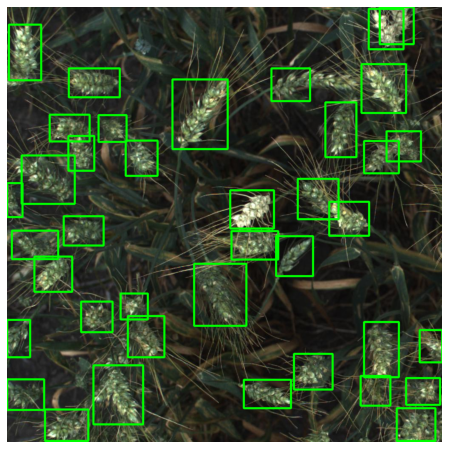


Đánh giá mô hình trên tập valid



Đánh giá mô hình trên tập test



2.2.3 Test mô hình

Plot bouding box trên ảnh của tập test bên trái (bouding box được cung cấp từ dữ liệu); bên phải (sử dụng bounding box được dự đoán từ model đã train)

Chi tiết của việc train model được thể hiện trong file Faster\_RCNN.ipynb.

1. Cài đặt Mask R-CNN
   1. Dataset

Sử dụng dữ liệu từ Penn-Fudan Database for Pedestrian Detection and Segmentation về người đi bộ.

Bộ dữ liệu gồm 170 hình ảnh với 345 người đi bộ được gắn nhãn.

* 1. Cách cài đặt (sử dụng pytorch)
     1. Chuẩn bị dữ liệu

Download dữ liệu và giải nén.



Tương tự như Faster R-CNN model này cũng nhận cặp (x\_train, y\_train) theo định dạng(image, target).

* Tạo class PennFudanDataset để đưa dữ liệu về dạng phù hợp với mô hình.

class PennFudanDataset(torch.utils.data.Dataset):

    def \_\_init\_\_(self, root, transforms=None):

        self.root = root

        self.transforms = transforms

        # load all image files, sorting them to

        # ensure that they are aligned

        self.imgs = list(sorted(os.listdir(os.path.join(root, "PNGImages"))))

        self.masks = list(sorted(os.listdir(os.path.join(root, "PedMasks"))))

    def \_\_getitem\_\_(self, idx):

        # load images ad masks

        img\_path = os.path.join(self.root, "PNGImages", self.imgs[idx])

        mask\_path = os.path.join(self.root, "PedMasks", self.masks[idx])

        img = Image.open(img\_path).convert("RGB")

        # note that we haven't converted the mask to RGB,

        # because each color corresponds to a different instance

        # with 0 being background

        mask = Image.open(mask\_path)

        mask = np.array(mask)

        # instances are encoded as different colors

        obj\_ids = np.unique(mask)

        # first id is the background, so remove it

        obj\_ids = obj\_ids[1:]

        # split the color-encoded mask into a set

        # of binary masks

        masks = mask == obj\_ids[:, None, None]

        # get bounding box coordinates for each mask

        num\_objs = len(obj\_ids)

        boxes = []

        for i in range(num\_objs):

            pos = np.where(masks[i])

            xmin = np.min(pos[1])

            xmax = np.max(pos[1])

            ymin = np.min(pos[0])

            ymax = np.max(pos[0])

            boxes.append([xmin, ymin, xmax, ymax])

        boxes = torch.as\_tensor(boxes, dtype=torch.float32)

        # there is only one class

        labels = torch.ones((num\_objs,), dtype=torch.int64)

        masks = torch.as\_tensor(masks, dtype=torch.uint8)

        image\_id = torch.tensor([idx])

        area = (boxes[:, 3] - boxes[:, 1]) \* (boxes[:, 2] - boxes[:, 0])

        # suppose all instances are not crowd

        iscrowd = torch.zeros((num\_objs,), dtype=torch.int64)

        target = {}

        target["boxes"] = boxes

        target["labels"] = labels

        target["masks"] = masks

        target["image\_id"] = image\_id

        target["area"] = area

        target["iscrowd"] = iscrowd

        if self.transforms is not None:

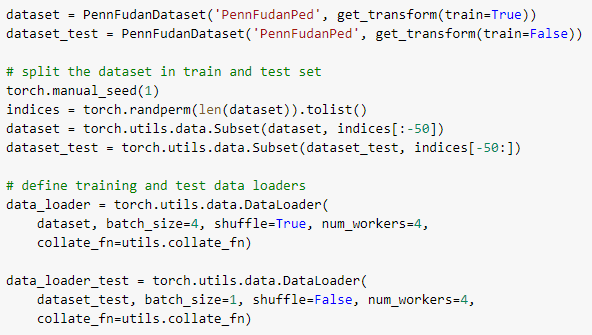
            img, target = self.transforms(img, target)

        return img, target

    def \_\_len\_\_(self):

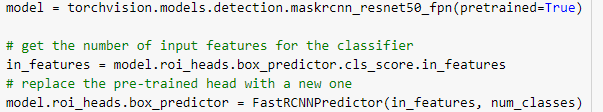
        return len(self.imgs)

Khởi tạo dataset (đưa dữ liệu về dạng phù hợp với mô hình) và dataloader để chuẩn bị cho việc train mô hình.

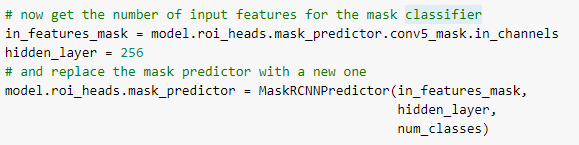


* + 1. Train model

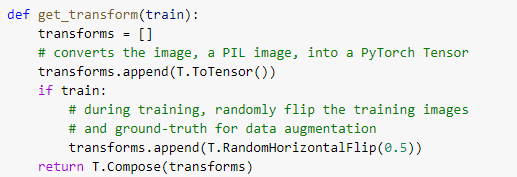
Load pre-trained model của Mask R-CNN (tương tự như Faster R-CNN)



Nhưng do Mask R-CNN sẽ tạo mask cho hình ảnh => load thêm phần mask.



Chuyển hình ảnh về dạng Pytorch tensor.



Train model với 10 epoch, sử dụng optimizer là SGD với learning rate là 0.005 và momentum là 0.9. Sau mỗi 3 epoch learning rate sẽ giảm 10 lần.

* + 1. Test mô hình

Kết quả thu được từ mô hình trên ảnh thuôc tập test.



Dự đoán được 15 bounding box từ hình ảnh được đưa vào. Những hình ảnh được thể hiện ở trên khá tốt nhưng có một vài bounding box:



Chi tiết của việc train model được thể hiện trong file Mask\_R\_CNN.ipynb

1. So sánh thời gian chạy của Faster R-CNN và Mask R-CNN

|  |  |  |
| --- | --- | --- |
| Số lượng object | Faster R-CNN | Mask R-CNN |
| 2 | 3475.208759 ms | 813.726902 ms |
| 4 | 778.364420 ms | 822.936058 ms |
| >10 | 842.340469 ms | 899.846315 ms |
| Video (100 frame) | 22467.616081 ms | 27021.537066 ms |

Thời gian chạy của mỗi frame khoảng 0.22s (Faster R-CNN)

Thời gian chạy của mỗi frame khoảng 0.27s (Mask R-CNN) (Mask R-CNN nhận diện được nhiều object hơn Faster R-CNN nên khi để cùng một ngưỡng (0.5) thì thời gian chạy lâu hơn)