Faster R-CNN Approach

Faster R-CNN

Import Libraries

```
# Nedded Libraries

# PyTorch
import torch
import torchvision
from torch.utils.data import Dataset, DataLoader
from torchvision import models
from torchvision.models.detection.faster_rcnn import
FastRCNNPredictor, fasterrcnn_resnet50_fpn
import albumentations as A

# Image processing
from PIL import Image, ImageDraw, ExifTags, ImageColor, ImageFont

# Image Plots
from matplotlib import pyplot as plt
import matplotlib.patches as patches

# Data managements
import numpy as np
import pandas as pd

# File interpretation
import os
import xml.etree.ElementTree as ET
import random

# Others
import time
from collections import Counter
from random import seed, randint
from datetime import datetime
```

Find Annotation Files

Indicate the path for the annotation files.

```
# Annotations directory path
ann_directory = '/content/FaceMaskDetection/annotations'

# List directory
ann_files = os.listdir(ann_directory)
```

Find Image Files

Indicate the path for the image files.

```
# Image directory path
img_directory = '/content/FaceMaskDetection/images'

# List directory
img_files = os.listdir(img_directory)
```

Find Annotation Files

Indicate the path for the annotation files.

Helper Functions

This are auxiliary functions used throughout the notebook. This way the notebook stays tidy and clean.

```
def draw_bounding_boxes(img_tensor, target=None, prediction=None):
    """Draws bounding boxes in given images. Displays them

        Inputs:
          img:
            Image in tensor format.
          target:
            target dictionary containing bboxes list wit format ->
[xmin, ymin, xmax, ymax]

        Returns:
          None
        """

    img = torchvision.transforms.ToPILImage()(img_tensor)

    # fetching the dimensions
    wid, hgt = img.size
    print(str(wid) + "x" + str(hgt))

    # Img to draw in
    draw = ImageDraw.Draw(img)

    if target:
        target_bboxes = target['boxes'].numpy().tolist()
        target_labels = decode_labels(target['labels'].numpy())

        for i in range(len(target_bboxes)):
            # Create Rectangle patches and add the patches to the axes
            draw.rectangle(target_bboxes[i], fill=None,
outline='green', width=2)
```

```
            draw.text(target_bboxes[i][:2], target_labels[i],
fill='green', font=None, anchor=None, spacing=4,
                      align='left', direction=None, features=None,
language=None, stroke_width=0, stroke_fill=None,
                      embedded_color=False)

    if prediction:
        prediction_bboxes =
prediction['boxes'].detach().cpu().numpy().tolist()
        prediction_labels =
decode_labels(prediction['labels'].detach().cpu().numpy())
        for i in range(len(prediction_bboxes)):
            # Create Rectangle patches and add the patches to the axes
            draw.rectangle(prediction_bboxes[i], fill=None,
outline='red', width=2)
            draw.text(prediction_bboxes[i][:2], prediction_labels[i],
fill='red', font=None, anchor=None, spacing=4,
                      align='left', direction=None, features=None,
language=None, stroke_width=0, stroke_fill=None,
                      embedded_color=False)

    display(img)

def encoded_labels(lst_labels):
    """Encodes label classes from string to integers.

        Labels are encoded accordingly:
            - background => 0
            - with_mask => 1
            - mask_weared_incorrect => 2
            - without_mask => 3

            Args:
              lst_labels:
                A list with classes in string format (e.g.
['with_mask', 'mask_weared_incorrect'...]).

            Returns:
              encoded:
                A list with integers that represent each class.
            """

    encoded=[]
    for label in lst_labels:
        if label == "with_mask":
            code = 1
        elif label == "mask_weared_incorrect":
            code = 2
        elif label == "without_mask":
            code = 3
```

```
        else:
            code = 0
        encoded.append(code)
    return encoded

def decode_labels(lst_labels):
    """
    Decode label classes from integers to strings.
    Labels are encoded accordingly:
        - background => 0
        - with_mask => 1
        - mask_weared_incorrect => 2
        - without_mask => 3

    Args:
      lst_labels:
        A list with classes in integer format (e.g. [1, 2, ...]).

    Returns:
        A list with strings that represent each class.
    """

    labels=[]
    for code in lst_labels:
        if code == 1:
            label = "with_mask"
        elif code == 2:
            label = "mask_weared_incorrect"
        elif code == 3:
            label = "without_mask"
        else:
            label = 'background'
        labels.append(label)
    return labels

def build_model(nclasses):
    """
    Builds model. Uses Faster R-CNN pre-trained on COCO dataset.

    Args:
      nclasses:
        number of classes

    Return:
      model: Faster R-CNN pre-trained model
    """
    # load pre-trained model on COCO
    model = fasterrcnn_resnet50_fpn(pretrained=True, min_size=400,
max_size=700)
```

```
    # get the number of input features for the classifier
    in_features = model.roi_heads.box_predictor.cls_score.in_features

    # replace the pre-trained head with a new one
    model.roi_heads.box_predictor = FastRCNNPredictor(in_features,
nclasses)

    return model

def train_model(model, loader, optimizer, scheduler, epochs, device):
  """
    Inputs:
      - model
      - loader: Dataloader PyTorch object with training data
      - optimizer
      - scheduler
      - epochs
      - device

    Returns:
      - model
      - loss_list: list with mean loss per epoch. Epoch 1 is in idex
0.
    """
  # Create a loss list to keep epoch average loss
  loss_list = []
  # Epochs
  for epoch in range(epochs):
      print('Starting epoch...... {}/{} '.format(epoch + 1, epochs))
      iteration = 0
      loss_sub_list = []
      start = time.time()
      for images, targets in loader:
          # Agregate images in batch loader
          images = list(image.to(device) for image in images)

          # Agregate targets in batch loader
          targets = [{key: val.to(device) for key, val in
target.items()} for target in targets]

          # Sets model to train mode (just a flag)
          model.train()

          # Output of model returns loss and detections
          optimizer.zero_grad()
          output = model(images, targets)

          # Calculate Cost
          losses = sum(loss for loss in output.values())
          loss_value = losses.item()
```

```python
            loss_sub_list.append(loss_value)
            print('')

            # Update optimizer and learning rate
            losses.backward()
            optimizer.step()
            iteration += 1
            print('Iteration: {:d} --> Loss: {:.3f}'.format(iteration,
loss_value))

        end = time.time()
        # update scheduler
        scheduler.step()
        # print the loss of epoch
        epoch_loss = np.mean(loss_sub_list)
        loss_list.append(epoch_loss)
        print('Epoch loss: {:.3f} , time used:
({:.1f}s)'.format(epoch_loss, end - start))

  return model, loss_list

def apply_nms(orig_prediction, iou_thresh):
    """
    Applies non max supression and eliminates low score bounding
boxes.

      Args:
        orig_prediction: the model output. A dictionary containing
element scores and boxes.
        iou_thresh: Intersection over Union threshold. Every bbox
prediction with an IoU greater than this value
                      gets deleted in NMS.

      Returns:
        final_prediction: Resulting prediction
    """

    # torchvision returns the indices of the bboxes to keep
    keep = torchvision.ops.nms(orig_prediction['boxes'],
orig_prediction['scores'], iou_thresh)

    # Keep indices from nms
    final_prediction = orig_prediction
    final_prediction['boxes'] = final_prediction['boxes'][keep]
    final_prediction['scores'] = final_prediction['scores'][keep]
    final_prediction['labels'] = final_prediction['labels'][keep]

    return final_prediction
```

```python
def remove_low_score_bb(orig_prediction, score_thresh):
    """
    Eliminates low score bounding boxes.

    Args:
        orig_prediction: the model output. A dictionary containing
element scores and boxes.
        score_thresh: Boxes with a lower confidence score than this
value get deleted

    Returns:
        final_prediction: Resulting prediction
    """

    # Remove low confidence scores according to given threshold
    index_list_scores = []
    scores = orig_prediction['scores'].detach().cpu().numpy()
    for i in range(len(scores)):
        if scores[i] > score_thresh:
            index_list_scores.append(i)
    keep = torch.tensor(index_list_scores)

    # Keep indices from high score bb
    final_prediction = orig_prediction
    final_prediction['boxes'] = final_prediction['boxes'][keep]
    final_prediction['scores'] = final_prediction['scores'][keep]
    final_prediction['labels'] = final_prediction['labels'][keep]

    return final_prediction

def collate_fn(batch):
    # Collate function for Dataloader
    return tuple(zip(*batch))

def IOU(box1, box2):
    '''
    Intersection over Union - IoU
    *------------
    |   (x2min,y2min)
    |   *----------
    |   | ######| |
    ----|------* (x1max,y1max)
        |         |
        ----------

    Args:
        box1: [xmin,ymin,xmax,ymax]
        box2: [xmin,ymin,xmax,ymax]

    Returns:
```

```
        iou -> value of intersection over union of the 2 boxes

    '''

    # Compute coordinates of intersection
    xmin_inter = max(box1[0], box2[0])
    ymin_inter = max(box1[1], box2[1])
    xmax_inter = min(box1[2], box2[2])
    ymax_inter = min(box1[3], box2[3])

    # calculate area of intersection rectangle
    inter_area = max(0, xmax_inter - xmin_inter + 1) * max(0,
ymax_inter - ymin_inter + 1) # FIXME why plus one?

    # calculate boxes areas
    area1 = (box1[2] - box1[0] + 1) * (box1[3] - box1[1] + 1)
    area2 = (box2[2] - box2[0] + 1) * (box2[3] - box2[1] + 1)

    # compute IoU
    iou = inter_area / float(area1 + area2 - inter_area)
    assert iou >= 0
    return iou

def compute_AP(ground_truth, predictions, iou_thresh=0.5,
n_classes=4):
    """
    Calculates Average Precision across all classes.

    Args:
        ground_truth: list with ground-truth objects. Needs to have
the following format: [sequence, frame, obj, [xmin, ymin, xmax, ymax],
label, score]
        predictions: list with predictions objects. Needs to have the
following format: [sequence, frame, obj, [xmin, ymin, xmax, ymax],
label, score]
        iou_thresh: IoU to which a prediction compared to a ground-
truth is considered right.
        n_classes: number of existent classes

    Returns:
        Average precision for the specified threshold.
    """
    # Initialize lists
    APs = []
    class_gt = []
    class_predictions = []

    # AP is computed for each class
    for c in range(n_classes):
        # Find gt and predictions of the class
```

```
        for gt in ground_truth:
            if gt[4] == c:
                class_gt.append(gt)
        for predict in predictions:
            if predict[4] == c:
                class_predictions.append(predict)

        # Create dict with array of zeros for bb in each image
        gt_amount_bb = Counter([gt[1] for gt in class_gt])
        for key, val in gt_amount_bb.items():
            gt_amount_bb[key] = np.zeros(val)

        # Sort class predictions by their score
        class_predictions = sorted(class_predictions, key=lambda x:
x[5], reverse=True)

        # Create arrays for Positives (True and False)
        TP = np.zeros(len(class_predictions))
        FP = np.zeros(len(class_predictions))
        # Number of true boxes
        truth = len(class_gt)

        # Initializing aux variables
        epsilon = 1e-6

        # Iterate over predictions in each image and compare with
ground truth
        for predict_idx, prediction in enumerate(class_predictions):
            # Filter prediction image ground truths
            image_gt = [obj for obj in class_gt if obj[1] ==
prediction[1]]

            # Initializing aux variables
            best_iou = -1
            best_gt_iou_idx = -1

            # Iterate through image ground truths and calculate IoUs
            for gt_idx, gt in enumerate(image_gt):
                iou = IOU(prediction[3], gt[3])
                if iou > best_iou:
                    best_iou = iou
                    best_gt_iou_idx = gt_idx

            # If the best IoU is greater that thresh than an TP
prediction has been found
            if best_iou > iou_thresh and best_gt_iou_idx > -1:
                # Check if gt box was already covered
                if  gt_amount_bb[prediction[1]][best_gt_iou_idx] == 0:
                    gt_amount_bb[prediction[1]][best_gt_iou_idx] = 1
```

```
# set as covered
                    TP[predict_idx] = 1  # Count as true positive
                else:
                    FP[predict_idx] = 1
            else:
                FP[predict_idx] = 1

        # Calculate recall and precision
        TP_cumsum = np.cumsum(TP)
        FP_cumsum = np.cumsum(FP)
        recall = np.append([0], TP_cumsum / (truth + epsilon))
        precision = np.append([1], np.divide(TP_cumsum, (TP_cumsum +
FP_cumsum + epsilon)))

        # Calculate the area precision/recall and add to list
        APs.append(np.trapz(precision, recall))

    return sum(APs)/len(APs)  # average of class precisions


def compute_mAP(ground_truth, predictions, n_classes):
    """
    Calls AP computation for different levels of IoUs, [0.5:.05:0.95].

    Args:
        ground_truth: list with ground-truth objects. Needs to have
the following format: [sequence, frame, obj, [xmin, ymin, xmax, ymax],
label, score]
        predictions: list with predictions objects. Needs to have the
following format: [sequence, frame, obj, [xmin, ymin, xmax, ymax],
label, score]
        n_classes: number of existent classes.

    Returns:
        mAp and list with APs for each IoU threshold.
    """
    # return mAP
    APs = [compute_AP(ground_truth, predictions, iou_thresh,
n_classes) for iou_thresh in np.arange(0.5, 1.0, 0.05)]
    return np.mean(APs), APs

@torch.no_grad()
def evaluate(model, data_loader, device, sequences=1):
    """
    Evaluates model mAP for IoU range of [0.5:.05:0.95].

    Args:
        model: -
        data_loader: -
        device: -
```

```
        sequences: the number of sequences of images to pass, if any

    Returns:
        mAP and AP list for each IoU threshold in range [0.5:.05:0.95]
    """

    # Set evaluation mode flag
    model.eval()
    # Create list with all object detection -> [set, frame, obj,
[xmin,ymin,xmax,ymax], label, score]
    ground_truth = []
    predictions = []

    # Gather all targets and outputs on test set
    for image, targets in data_loader:
        image = [img.to(device) for img in image]
        outputs = model(image)
        for idx in range(len(outputs)):
            outputs[idx] = apply_nms(outputs[idx], iou_thresh=0.5)

        # create list for targets and outputs to pass to compute_mAP()
        # lists have the following structure:  [sequence, frame,
obj_idx, [xmin, ymin, xmax, ymax], label, score]
        for s in range(sequences):
            obj_gt = 0
            obj_target = 0
            for out, target in zip(outputs, targets):

                for i in range(len(target['boxes'])):
                    ground_truth.append([s,
target['image_id'].detach().cpu().numpy()[0], obj_target,

target['boxes'].detach().cpu().numpy()[i],

target['labels'].detach().cpu().numpy()[i], 1])
                    obj_target += 1

                for j in range(len(out['boxes'])):
                    predictions.append([s,
target['image_id'].detach().cpu().numpy()[0], obj_gt,

out['boxes'].detach().cpu().numpy()[j],

out['labels'].detach().cpu().numpy()[j],

out['scores'].detach().cpu().numpy()[j]])
                    obj_gt += 1

    mAP, AP = compute_mAP(ground_truth, predictions, n_classes=4)
```

```
    print("mAP:{:.3f}".format(mAP))
    for ap_metric, iou in zip(AP, np.arange(0.5, 1, 0.05)):
        print("\tAP at IoU level [{:.2f}]: {:.3f}".format(iou, 
ap_metric))

    return mAP, AP
```

Create Dataset Class

Dataset class to feed the dataloader.

```
# Create dataset object
class MyDataset(Dataset):

    # Constructor
    def __init__(self, ann_dir, img_dir, transform=None, 
mode='train'):

        # Image directories
        self.ann_dir = ann_dir
        self.img_dir = img_dir

        # The transform is goint to be used on image
        self.transform = transform

        # Create dataframe to hold info
        self.data = pd.DataFrame(columns=['Filename', 'BoundingBoxes', 
'Labels', 'Area', 'N_Objects'])

        # Append rows with image filename and respective bounding 
boxes to the df
        for file in enumerate(os.listdir(img_dir)):

            # Find image annotation file
            ann_file_path = os.path.join(ann_dir, file[1][:-4]) + 
'.xml'

            # Read XML file and return bounding boxes and class 
attributes
            objects = self.read_XML_classf(ann_file_path)

            # Create list of labels in an image
            list_labels = encoded_labels(objects[0]['labels'])

            # Create list of bounding boxes in an image
            list_bb = []
            list_area = []
            n_obj = len(objects[0]['objects'])
            for i in objects[0]['objects']:
                list = [i['xmin'], i['ymin'], i['xmax'], i['ymax']]
```

```
                list_bb.append(list)
                list_area.append((i['xmax'] - i['xmin']) * (i['ymax']
- i['ymin']))

            # Create dataframe object with row containing [(Image file
name),(Bounding Box List)]
            df = pd.DataFrame([[file[1], list_bb, list_labels,
list_area, n_obj]],
                              columns=['Filename', 'BoundingBoxes',
'Labels', 'Area', 'N_Objects'])
            self.data = self.data.append(df)

        if mode == 'train':
            self.data = self.data[:680]
        elif mode == 'validation':
            self.data = self.data[680:700]
        elif mode == 'test':
            self.data = self.data[700:850]

        # Number of images in dataset
        self.len = self.data.shape[0]

        # Get the length

    def __len__(self):
        return self.len

    # Getter
    def __getitem__(self, idx):

        # Image file path
        img_name = os.path.join(self.img_dir, self.data.iloc[idx, 0])

        # Open image file and tranform to tensor
        img = Image.open(img_name).convert('RGB')

        # Get bounding box coordinates
        bbox = torch.tensor(self.data.iloc[idx, 1])

        # Get labels
        labels = torch.tensor(self.data.iloc[idx, 2])

        # Get bounding box areas
        area = torch.tensor(self.data.iloc[idx, 3])

        # If any, aplly tranformations to image and bounding box mask
        if self.transform:
            # Convert PIL image to numpy array
            img = np.array(img)
```

```
            # Apply transformations
            transformed = self.transform(image=img, bboxes=bbox)
            # Convert numpy array to PIL Image
            img = Image.fromarray(transformed['image'])
            # Get transformed bb
            bbox = torch.tensor(transformed['bboxes'])

        # suppose all instances are not crowd
        num_objs = self.data.iloc[idx, 4]
        iscrowd = torch.zeros((num_objs,), dtype=torch.int64)

        # Transform img to tensor
        img = torchvision.transforms.ToTensor()(img)

        # Build Targer dict
        target= {"boxes": bbox, "labels": labels, "image_id":
torch.tensor([idx]), "area": area, "iscrowd": iscrowd}

        return img, target

    # XML reader -> returns dictionary with image bounding boxes sizes
    def read_XML_classf(self, ann_file_path):
        bboxes = [{
            'file': ann_file_path,
            'labels': [],
            'objects': []
        }]

        # Reading XML file objects and print Bounding Boxes
        tree = ET.parse(ann_file_path)
        root = tree.getroot()
        objects = root.findall('object')

        for obj in objects:
            # label
            label = obj.find('name').text
            bboxes[0]['labels'].append(label)

            # bbox dimensions
            bndbox = obj.find('bndbox')
            xmin = int(bndbox.find('xmin').text)
            ymin = int(bndbox.find('ymin').text)
            xmax = int(bndbox.find('xmax').text)
            ymax = int(bndbox.find('ymax').text)
            bboxes[0]['objects'].append({'xmin': xmin, 'ymin': ymin,
'xmax': xmax, 'ymax': ymax})

        return bboxes
```

Create Data Pipeline

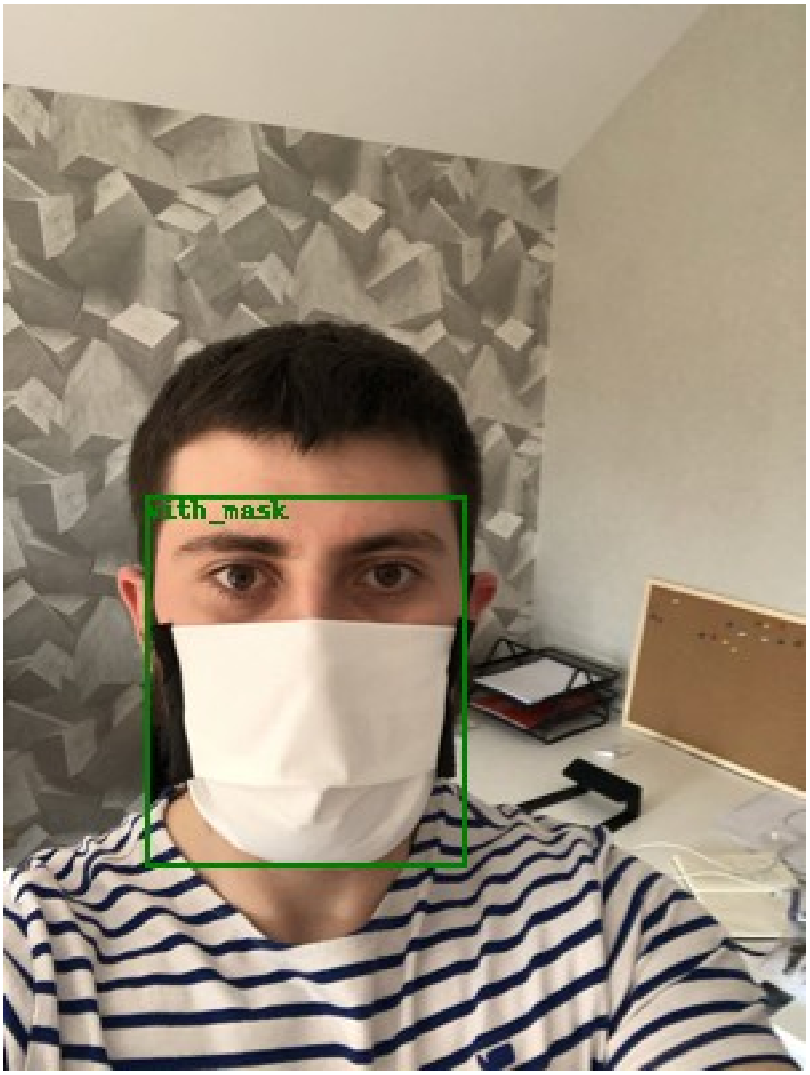
```
# Create Data Pipeline

# Training Data
dataset_train = MyDataset(ann_directory,img_directory, mode = 'train')
loader_train = DataLoader(dataset_train, batch_size=4, shuffle=True,
collate_fn=collate_fn)
# Validation Data
dataset_validation = MyDataset(ann_directory,img_directory, mode =
'validation')
loader_val = DataLoader(dataset_validation, batch_size=4,
shuffle=True, collate_fn=collate_fn)
# Test Data
dataset_test = MyDataset(ann_directory,img_directory, mode = 'test')
loader_test = DataLoader(dataset_test, batch_size=4, shuffle=True,
collate_fn=collate_fn)
```

Test if dataset is working correctly. Print out ground truth bounding box of first image.

```
# pick one image from the train set
img, target = dataset_train[0]
draw_bounding_boxes(img, target)

img, target = dataset_train[4]
draw_bounding_boxes(img, target)

img, target = dataset_train[7]
draw_bounding_boxes(img, target)
```

```
400x210
```

301x400



267x400

Setting up the Faster R-CNN Model

```
# Setting up GPU device
device = torch.device('cuda') if torch.cuda.is_available() else
torch.device('cpu')

# Nº of classes: background, with_mask, mask_weared_incorrect,
without_mask and build model (faster r-cnn)
num_classes = 4
model = build_model(num_classes)

model = model.to(device)
```

```
/usr/local/lib/python3.7/dist-packages/torchvision/models/
_utils.py:209: UserWarning: The parameter 'pretrained' is deprecated
since 0.13 and will be removed in 0.15, please use 'weights' instead.
  f"The parameter '{pretrained_param}' is deprecated since 0.13 and
will be removed in 0.15, "
/usr/local/lib/python3.7/dist-packages/torchvision/models/_utils.py:22
```

```
3: UserWarning: Arguments other than a weight enum or `None` for
'weights' are deprecated since 0.13 and will be removed in 0.15. The
current behavior is equivalent to passing
`weights=FasterRCNN_ResNet50_FPN_Weights.COCO_V1`. You can also use
`weights=FasterRCNN_ResNet50_FPN_Weights.DEFAULT` to get the most up-
to-date weights.
  warnings.warn(msg)
Downloading:
"https://download.pytorch.org/models/fasterrcnn_resnet50_fpn_coco-
258fb6c6.pth" to
/root/.cache/torch/hub/checkpoints/fasterrcnn_resnet50_fpn_coco-
258fb6c6.pth
```

{"version_major":2,"version_minor":0,"model_id":"5d22a8e2c3dd4234b40ea
4f6e7a9dc3a"}

```python
# Set Hyper-parameters

# Network params
params = [p for p in model.parameters() if p.requires_grad]

# Optimizers
optimizer = torch.optim.Adam(params, lr=0.0001)
#optimizer = torch.optim.SGD(params, lr=0.005)

# Learning Rate, lr decreases to half every 2 epochs
lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=2,
gamma=0.5)

# Number of epochs to perform
epochs=20
```

Train the model

```
pip install
git+https://github.com/gautamchitnis/cocoapi.git@cocodataset-
master#subdirectory=PythonAPI
```

```
Looking in indexes: https://pypi.org/simple, https://us-
python.pkg.dev/colab-wheels/public/simple/
Collecting
git+https://github.com/gautamchitnis/cocoapi.git@cocodataset-
master#subdirectory=PythonAPI
  Cloning https://github.com/gautamchitnis/cocoapi.git (to revision
cocodataset-master) to /tmp/pip-req-build-wzji15jf
  Running command git clone -q
https://github.com/gautamchitnis/cocoapi.git /tmp/pip-req-build-
wzji15jf
  Running command git checkout -b cocodataset-master --track
origin/cocodataset-master
  Switched to a new branch 'cocodataset-master'
```

```
  Branch 'cocodataset-master' set up to track remote branch
'cocodataset-master' from 'origin'.
Building wheels for collected packages: pycocotools
  Building wheel for pycocotools (setup.py) ... e=pycocotools-2.0-
cp37-cp37m-linux_x86_64.whl size=265320
sha256=3aba1e2d4e84d0edca62fc1d4cf960c07e7dea2d21a5d64744e0b3e85b3a979
6
  Stored in directory:
/tmp/pip-ephem-wheel-cache-vs71d2py/wheels/6e/c9/59/56484d4d5ac1ab292a
452b4c3870277256551505954fc4a1db
Successfully built pycocotools
Installing collected packages: pycocotools
  Attempting uninstall: pycocotools
    Found existing installation: pycocotools 2.0.4
    Uninstalling pycocotools-2.0.4:
      Successfully uninstalled pycocotools-2.0.4
Successfully installed pycocotools-2.0
```

```
!git clone https://github.com/pytorch/vision.git
%cd vision
!git checkout v0.3.0

!cp references/detection/utils.py ../
!cp references/detection/transforms.py ../
!cp references/detection/coco_eval.py ../
!cp references/detection/engine.py ../
!cp references/detection/coco_utils.py ../
%cd ..
```

```
Cloning into 'vision'...
remote: Enumerating objects: 190176, done.ote: Counting objects: 100%
(33/33), done.ote: Compressing objects: 100% (26/26), done.ote: Total
190176 (delta 11), reused 12 (delta 7), pack-reused 190143ake
experimental
changes and commit them, and you can discard any commits you make in
this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you
may
do so (now or later) by using -b with the checkout command again.
Example:

  git checkout -b <new-branch-name>

HEAD is now at be376084d version check against PyTorch's CUDA version
/content
```

```
from engine import train_one_epoch
# Training
```

```python
for epoch in range(epochs):
    # train for one epoch, printing every 50 iterations
    train_one_epoch(model, optimizer, loader_train, device, epoch,
print_freq=20)
    # update the learning rate
    lr_scheduler.step()
    # evaluate on the test dataset
    evaluate(model, loader_val, device=device)
```

```
Epoch: [0]  [  0/170]  eta: 0:02:57  lr: 0.000001  loss: 2.1042
(2.1042)  loss_classifier: 1.6616 (1.6616)  loss_box_reg: 0.0811
(0.0811)  loss_objectness: 0.2105 (0.2105)  loss_rpn_box_reg: 0.1509
(0.1509)  time: 1.0424  data: 0.1549  max mem: 1391
Epoch: [0]  [ 20/170]  eta: 0:01:06  lr: 0.000013  loss: 1.7572
(1.6751)  loss_classifier: 1.2356 (1.2573)  loss_box_reg: 0.1703
(0.1726)  loss_objectness: 0.1480 (0.1799)  loss_rpn_box_reg: 0.0220
(0.0654)  time: 0.4127  data: 0.1111  max mem: 2507
Epoch: [0]  [ 40/170]  eta: 0:00:55  lr: 0.000024  loss: 0.7379
(1.2276)  loss_classifier: 0.3061 (0.8143)  loss_box_reg: 0.3008
(0.2367)  loss_objectness: 0.0648 (0.1258)  loss_rpn_box_reg: 0.0233
(0.0508)  time: 0.4027  data: 0.1004  max mem: 2507
Epoch: [0]  [ 60/170]  eta: 0:00:45  lr: 0.000036  loss: 0.4773
(1.0015)  loss_classifier: 0.1737 (0.6075)  loss_box_reg: 0.2569
(0.2530)  loss_objectness: 0.0275 (0.0975)  loss_rpn_box_reg: 0.0105
(0.0435)  time: 0.4028  data: 0.1013  max mem: 2507
Epoch: [0]  [ 80/170]  eta: 0:00:37  lr: 0.000048  loss: 0.4473
(0.8779)  loss_classifier: 0.1340 (0.4972)  loss_box_reg: 0.2861
(0.2585)  loss_objectness: 0.0280 (0.0830)  loss_rpn_box_reg: 0.0167
(0.0391)  time: 0.4163  data: 0.1110  max mem: 2697
Epoch: [0]  [100/170]  eta: 0:00:28  lr: 0.000060  loss: 0.3520
(0.7831)  loss_classifier: 0.1166 (0.4248)  loss_box_reg: 0.2058
(0.2511)  loss_objectness: 0.0146 (0.0723)  loss_rpn_box_reg: 0.0065
(0.0348)  time: 0.4034  data: 0.1049  max mem: 2697
Epoch: [0]  [120/170]  eta: 0:00:20  lr: 0.000072  loss: 0.4257
(0.7362)  loss_classifier: 0.1289 (0.3801)  loss_box_reg: 0.2208
(0.2515)  loss_objectness: 0.0322 (0.0687)  loss_rpn_box_reg: 0.0236
(0.0359)  time: 0.4051  data: 0.1056  max mem: 2697
Epoch: [0]  [140/170]  eta: 0:00:12  lr: 0.000083  loss: 0.4665
(0.6990)  loss_classifier: 0.1249 (0.3479)  loss_box_reg: 0.2524
(0.2508)  loss_objectness: 0.0144 (0.0653)  loss_rpn_box_reg: 0.0062
(0.0349)  time: 0.4090  data: 0.1078  max mem: 2697
Epoch: [0]  [160/170]  eta: 0:00:04  lr: 0.000095  loss: 0.3958
(0.6606)  loss_classifier: 0.1081 (0.3204)  loss_box_reg: 0.2100
(0.2466)  loss_objectness: 0.0151 (0.0599)  loss_rpn_box_reg: 0.0079
(0.0337)  time: 0.4144  data: 0.1150  max mem: 2697
Epoch: [0]  [169/170]  eta: 0:00:00  lr: 0.000100  loss: 0.3870
(0.6412)  loss_classifier: 0.1127 (0.3085)  loss_box_reg: 0.2100
(0.2428)  loss_objectness: 0.0113 (0.0573)  loss_rpn_box_reg: 0.0127
(0.0326)  time: 0.4277  data: 0.1202  max mem: 2794
Epoch: [0] Total time: 0:01:10 (0.4141 s / it)
```

```
mAP:0.390
     AP at IoU level [0.50]: 0.666
     AP at IoU level [0.55]: 0.666
     AP at IoU level [0.60]: 0.632
     AP at IoU level [0.65]: 0.608
     AP at IoU level [0.70]: 0.549
     AP at IoU level [0.75]: 0.479
     AP at IoU level [0.80]: 0.226
     AP at IoU level [0.85]: 0.060
     AP at IoU level [0.90]: 0.017
     AP at IoU level [0.95]: 0.000
Epoch: [1]  [  0/170]  eta: 0:00:57  lr: 0.000100  loss: 0.4972
(0.4972)  loss_classifier: 0.1732 (0.1732)  loss_box_reg: 0.2865
(0.2865)  loss_objectness: 0.0262 (0.0262)  loss_rpn_box_reg: 0.0114
(0.0114)  time: 0.3408  data: 0.0574  max mem: 2794
Epoch: [1]  [ 20/170]  eta: 0:00:54  lr: 0.000100  loss: 0.2797
(0.2914)  loss_classifier: 0.0942 (0.0968)  loss_box_reg: 0.1649
(0.1723)  loss_objectness: 0.0071 (0.0112)  loss_rpn_box_reg: 0.0068
(0.0112)  time: 0.3628  data: 0.0580  max mem: 2794
Epoch: [1]  [ 40/170]  eta: 0:00:47  lr: 0.000100  loss: 0.3508
(0.3219)  loss_classifier: 0.0933 (0.1016)  loss_box_reg: 0.2163
(0.1905)  loss_objectness: 0.0141 (0.0132)  loss_rpn_box_reg: 0.0113
(0.0165)  time: 0.3645  data: 0.0596  max mem: 2794
Epoch: [1]  [ 60/170]  eta: 0:00:39  lr: 0.000100  loss: 0.3592
(0.3315)  loss_classifier: 0.1031 (0.1014)  loss_box_reg: 0.2025
(0.1971)  loss_objectness: 0.0129 (0.0136)  loss_rpn_box_reg: 0.0137
(0.0194)  time: 0.3613  data: 0.0630  max mem: 2794
Epoch: [1]  [ 80/170]  eta: 0:00:32  lr: 0.000100  loss: 0.3977
(0.3441)  loss_classifier: 0.1186 (0.1052)  loss_box_reg: 0.2242
(0.2050)  loss_objectness: 0.0078 (0.0141)  loss_rpn_box_reg: 0.0084
(0.0199)  time: 0.3697  data: 0.0637  max mem: 2794
Epoch: [1]  [100/170]  eta: 0:00:25  lr: 0.000100  loss: 0.3831
(0.3510)  loss_classifier: 0.1259 (0.1078)  loss_box_reg: 0.2064
(0.2076)  loss_objectness: 0.0197 (0.0148)  loss_rpn_box_reg: 0.0222
(0.0208)  time: 0.3629  data: 0.0638  max mem: 2794
Epoch: [1]  [120/170]  eta: 0:00:18  lr: 0.000100  loss: 0.2674
(0.3452)  loss_classifier: 0.0875 (0.1064)  loss_box_reg: 0.1748
(0.2031)  loss_objectness: 0.0060 (0.0150)  loss_rpn_box_reg: 0.0063
(0.0207)  time: 0.3721  data: 0.0601  max mem: 2794
Epoch: [1]  [140/170]  eta: 0:00:10  lr: 0.000100  loss: 0.2462
(0.3347)  loss_classifier: 0.0754 (0.1033)  loss_box_reg: 0.1627
(0.1971)  loss_objectness: 0.0050 (0.0143)  loss_rpn_box_reg: 0.0042
(0.0200)  time: 0.3575  data: 0.0575  max mem: 2794
Epoch: [1]  [160/170]  eta: 0:00:03  lr: 0.000100  loss: 0.3389
(0.3334)  loss_classifier: 0.1002 (0.1035)  loss_box_reg: 0.1809
(0.1963)  loss_objectness: 0.0060 (0.0140)  loss_rpn_box_reg: 0.0072
(0.0197)  time: 0.3634  data: 0.0663  max mem: 2794
Epoch: [1]  [169/170]  eta: 0:00:00  lr: 0.000100  loss: 0.2710
(0.3316)  loss_classifier: 0.0872 (0.1024)  loss_box_reg: 0.1671
(0.1956)  loss_objectness: 0.0082 (0.0140)  loss_rpn_box_reg: 0.0108
```

```
(0.0196)  time: 0.3661  data: 0.0653  max mem: 2794
Epoch: [1] Total time: 0:01:01 (0.3642 s / it)
mAP:0.426
     AP at IoU level [0.50]: 0.698
     AP at IoU level [0.55]: 0.675
     AP at IoU level [0.60]: 0.654
     AP at IoU level [0.65]: 0.642
     AP at IoU level [0.70]: 0.591
     AP at IoU level [0.75]: 0.532
     AP at IoU level [0.80]: 0.346
     AP at IoU level [0.85]: 0.120
     AP at IoU level [0.90]: 0.004
     AP at IoU level [0.95]: 0.000
Epoch: [2]  [  0/170]  eta: 0:01:03  lr: 0.000050  loss: 0.4140
(0.4140)  loss_classifier: 0.1041 (0.1041)  loss_box_reg: 0.2435
(0.2435)  loss_objectness: 0.0194 (0.0194)  loss_rpn_box_reg: 0.0470
(0.0470)  time: 0.3708  data: 0.0549  max mem: 2794
Epoch: [2]  [ 20/170]  eta: 0:00:53  lr: 0.000050  loss: 0.1858
(0.2431)  loss_classifier: 0.0538 (0.0775)  loss_box_reg: 0.1200
(0.1438)  loss_objectness: 0.0046 (0.0092)  loss_rpn_box_reg: 0.0042
(0.0127)  time: 0.3556  data: 0.0624  max mem: 2794
Epoch: [2]  [ 40/170]  eta: 0:00:46  lr: 0.000050  loss: 0.2768
(0.2706)  loss_classifier: 0.0788 (0.0814)  loss_box_reg: 0.1706
(0.1631)  loss_objectness: 0.0050 (0.0100)  loss_rpn_box_reg: 0.0082
(0.0161)  time: 0.3651  data: 0.0626  max mem: 2794
Epoch: [2]  [ 60/170]  eta: 0:00:40  lr: 0.000050  loss: 0.2187
(0.2577)  loss_classifier: 0.0596 (0.0795)  loss_box_reg: 0.1343
(0.1554)  loss_objectness: 0.0046 (0.0087)  loss_rpn_box_reg: 0.0044
(0.0141)  time: 0.3704  data: 0.0620  max mem: 2794
Epoch: [2]  [ 80/170]  eta: 0:00:32  lr: 0.000050  loss: 0.2351
(0.2553)  loss_classifier: 0.0701 (0.0788)  loss_box_reg: 0.1365
(0.1529)  loss_objectness: 0.0030 (0.0077)  loss_rpn_box_reg: 0.0058
(0.0158)  time: 0.3673  data: 0.0662  max mem: 2794
Epoch: [2]  [100/170]  eta: 0:00:25  lr: 0.000050  loss: 0.2451
(0.2517)  loss_classifier: 0.0620 (0.0765)  loss_box_reg: 0.1431
(0.1520)  loss_objectness: 0.0034 (0.0075)  loss_rpn_box_reg: 0.0099
(0.0157)  time: 0.3640  data: 0.0623  max mem: 2794
Epoch: [2]  [120/170]  eta: 0:00:18  lr: 0.000050  loss: 0.1962
(0.2487)  loss_classifier: 0.0550 (0.0754)  loss_box_reg: 0.1335
(0.1510)  loss_objectness: 0.0021 (0.0073)  loss_rpn_box_reg: 0.0043
(0.0150)  time: 0.3581  data: 0.0644  max mem: 2794
Epoch: [2]  [140/170]  eta: 0:00:10  lr: 0.000050  loss: 0.2371
(0.2492)  loss_classifier: 0.0730 (0.0751)  loss_box_reg: 0.1473
(0.1514)  loss_objectness: 0.0042 (0.0073)  loss_rpn_box_reg: 0.0055
(0.0154)  time: 0.3608  data: 0.0573  max mem: 2794
Epoch: [2]  [160/170]  eta: 0:00:03  lr: 0.000050  loss: 0.1850
(0.2435)  loss_classifier: 0.0500 (0.0733)  loss_box_reg: 0.1216
(0.1482)  loss_objectness: 0.0036 (0.0070)  loss_rpn_box_reg: 0.0041
(0.0150)  time: 0.3638  data: 0.0655  max mem: 2794
Epoch: [2]  [169/170]  eta: 0:00:00  lr: 0.000050  loss: 0.2194
```

```
(0.2432)  loss_classifier: 0.0689 (0.0742)  loss_box_reg: 0.1316
(0.1474)  loss_objectness: 0.0036 (0.0070)  loss_rpn_box_reg: 0.0036
(0.0146)  time: 0.3621  data: 0.0620  max mem: 2794
Epoch: [2] Total time: 0:01:01 (0.3632 s / it)
mAP:0.455
     AP at IoU level [0.50]: 0.710
     AP at IoU level [0.55]: 0.694
     AP at IoU level [0.60]: 0.680
     AP at IoU level [0.65]: 0.654
     AP at IoU level [0.70]: 0.634
     AP at IoU level [0.75]: 0.579
     AP at IoU level [0.80]: 0.428
     AP at IoU level [0.85]: 0.151
     AP at IoU level [0.90]: 0.020
     AP at IoU level [0.95]: 0.000
Epoch: [3]  [  0/170]  eta: 0:01:05  lr: 0.000050  loss: 0.3210
(0.3210)  loss_classifier: 0.0884 (0.0884)  loss_box_reg: 0.1770
(0.1770)  loss_objectness: 0.0354 (0.0354)  loss_rpn_box_reg: 0.0203
(0.0203)  time: 0.3869  data: 0.0680  max mem: 2794
Epoch: [3]  [ 20/170]  eta: 0:00:54  lr: 0.000050  loss: 0.1955
(0.2044)  loss_classifier: 0.0585 (0.0639)  loss_box_reg: 0.1187
(0.1210)  loss_objectness: 0.0039 (0.0080)  loss_rpn_box_reg: 0.0046
(0.0116)  time: 0.3602  data: 0.0600  max mem: 2794
Epoch: [3]  [ 40/170]  eta: 0:00:47  lr: 0.000050  loss: 0.1854
(0.2003)  loss_classifier: 0.0481 (0.0625)  loss_box_reg: 0.1341
(0.1216)  loss_objectness: 0.0014 (0.0059)  loss_rpn_box_reg: 0.0035
(0.0102)  time: 0.3660  data: 0.0597  max mem: 2794
Epoch: [3]  [ 60/170]  eta: 0:00:39  lr: 0.000050  loss: 0.2053
(0.2137)  loss_classifier: 0.0722 (0.0670)  loss_box_reg: 0.1224
(0.1277)  loss_objectness: 0.0025 (0.0064)  loss_rpn_box_reg: 0.0050
(0.0127)  time: 0.3599  data: 0.0635  max mem: 2794
Epoch: [3]  [ 80/170]  eta: 0:00:32  lr: 0.000050  loss: 0.2003
(0.2166)  loss_classifier: 0.0667 (0.0670)  loss_box_reg: 0.1350
(0.1294)  loss_objectness: 0.0009 (0.0057)  loss_rpn_box_reg: 0.0067
(0.0144)  time: 0.3748  data: 0.0629  max mem: 2794
Epoch: [3]  [100/170]  eta: 0:00:25  lr: 0.000050  loss: 0.1670
(0.2123)  loss_classifier: 0.0511 (0.0647)  loss_box_reg: 0.1104
(0.1283)  loss_objectness: 0.0019 (0.0059)  loss_rpn_box_reg: 0.0030
(0.0133)  time: 0.3570  data: 0.0602  max mem: 2794
Epoch: [3]  [120/170]  eta: 0:00:18  lr: 0.000050  loss: 0.1593
(0.2078)  loss_classifier: 0.0503 (0.0631)  loss_box_reg: 0.1078
(0.1265)  loss_objectness: 0.0023 (0.0054)  loss_rpn_box_reg: 0.0030
(0.0128)  time: 0.3671  data: 0.0584  max mem: 2794
Epoch: [3]  [140/170]  eta: 0:00:10  lr: 0.000050  loss: 0.2051
(0.2096)  loss_classifier: 0.0630 (0.0632)  loss_box_reg: 0.1269
(0.1278)  loss_objectness: 0.0051 (0.0057)  loss_rpn_box_reg: 0.0102
(0.0129)  time: 0.3494  data: 0.0605  max mem: 2794
Epoch: [3]  [160/170]  eta: 0:00:03  lr: 0.000050  loss: 0.1918
(0.2091)  loss_classifier: 0.0472 (0.0626)  loss_box_reg: 0.1245
(0.1280)  loss_objectness: 0.0037 (0.0056)  loss_rpn_box_reg: 0.0034
```

```
(0.0129)  time: 0.3685  data: 0.0652  max mem: 2794
Epoch: [3]  [169/170]  eta: 0:00:00  lr: 0.000050  loss: 0.1918
(0.2092)  loss_classifier: 0.0482 (0.0624)  loss_box_reg: 0.1245
(0.1286)  loss_objectness: 0.0015 (0.0055)  loss_rpn_box_reg: 0.0041
(0.0127)  time: 0.3639  data: 0.0650  max mem: 2794
Epoch: [3] Total time: 0:01:01 (0.3624 s / it)
mAP:0.452
     AP at IoU level [0.50]: 0.721
     AP at IoU level [0.55]: 0.706
     AP at IoU level [0.60]: 0.678
     AP at IoU level [0.65]: 0.653
     AP at IoU level [0.70]: 0.641
     AP at IoU level [0.75]: 0.513
     AP at IoU level [0.80]: 0.424
     AP at IoU level [0.85]: 0.162
     AP at IoU level [0.90]: 0.018
     AP at IoU level [0.95]: 0.000
Epoch: [4]  [  0/170]  eta: 0:00:58  lr: 0.000025  loss: 0.1469
(0.1469)  loss_classifier: 0.0314 (0.0314)  loss_box_reg: 0.0915
(0.0915)  loss_objectness: 0.0083 (0.0083)  loss_rpn_box_reg: 0.0157
(0.0157)  time: 0.3432  data: 0.0558  max mem: 2794
Epoch: [4]  [ 20/170]  eta: 0:00:55  lr: 0.000025  loss: 0.1493
(0.1583)  loss_classifier: 0.0425 (0.0469)  loss_box_reg: 0.0925
(0.0976)  loss_objectness: 0.0022 (0.0038)  loss_rpn_box_reg: 0.0037
(0.0100)  time: 0.3700  data: 0.0624  max mem: 2794
Epoch: [4]  [ 40/170]  eta: 0:00:47  lr: 0.000025  loss: 0.1700
(0.1665)  loss_classifier: 0.0458 (0.0494)  loss_box_reg: 0.1185
(0.1027)  loss_objectness: 0.0011 (0.0038)  loss_rpn_box_reg: 0.0038
(0.0106)  time: 0.3647  data: 0.0596  max mem: 2794
Epoch: [4]  [ 60/170]  eta: 0:00:40  lr: 0.000025  loss: 0.1507
(0.1669)  loss_classifier: 0.0484 (0.0489)  loss_box_reg: 0.0991
(0.1038)  loss_objectness: 0.0007 (0.0036)  loss_rpn_box_reg: 0.0028
(0.0107)  time: 0.3637  data: 0.0576  max mem: 2794
Epoch: [4]  [ 80/170]  eta: 0:00:32  lr: 0.000025  loss: 0.1570
(0.1703)  loss_classifier: 0.0456 (0.0493)  loss_box_reg: 0.0899
(0.1063)  loss_objectness: 0.0023 (0.0037)  loss_rpn_box_reg: 0.0057
(0.0110)  time: 0.3612  data: 0.0647  max mem: 2794
Epoch: [4]  [100/170]  eta: 0:00:25  lr: 0.000025  loss: 0.1572
(0.1695)  loss_classifier: 0.0417 (0.0492)  loss_box_reg: 0.0921
(0.1057)  loss_objectness: 0.0008 (0.0042)  loss_rpn_box_reg: 0.0033
(0.0104)  time: 0.3598  data: 0.0633  max mem: 2794
Epoch: [4]  [120/170]  eta: 0:00:18  lr: 0.000025  loss: 0.1439
(0.1689)  loss_classifier: 0.0473 (0.0499)  loss_box_reg: 0.0841
(0.1040)  loss_objectness: 0.0006 (0.0039)  loss_rpn_box_reg: 0.0029
(0.0112)  time: 0.3686  data: 0.0602  max mem: 2794
Epoch: [4]  [140/170]  eta: 0:00:10  lr: 0.000025  loss: 0.1519
(0.1677)  loss_classifier: 0.0419 (0.0490)  loss_box_reg: 0.1027
(0.1042)  loss_objectness: 0.0014 (0.0037)  loss_rpn_box_reg: 0.0058
(0.0109)  time: 0.3557  data: 0.0567  max mem: 2794
Epoch: [4]  [160/170]  eta: 0:00:03  lr: 0.000025  loss: 0.1326
```

```
(0.1675)  loss_classifier: 0.0358 (0.0485)  loss_box_reg: 0.0873
(0.1042)  loss_objectness: 0.0013 (0.0039)  loss_rpn_box_reg: 0.0050
(0.0109)  time: 0.3500  data: 0.0631  max mem: 2794
Epoch: [4]  [169/170]  eta: 0:00:00  lr: 0.000025  loss: 0.1396
(0.1679)  loss_classifier: 0.0391 (0.0486)  loss_box_reg: 0.0962
(0.1047)  loss_objectness: 0.0012 (0.0039)  loss_rpn_box_reg: 0.0034
(0.0108)  time: 0.3525  data: 0.0678  max mem: 2794
Epoch: [4] Total time: 0:01:01 (0.3612 s / it)
mAP:0.464
     AP at IoU level [0.50]: 0.713
     AP at IoU level [0.55]: 0.706
     AP at IoU level [0.60]: 0.693
     AP at IoU level [0.65]: 0.659
     AP at IoU level [0.70]: 0.636
     AP at IoU level [0.75]: 0.526
     AP at IoU level [0.80]: 0.443
     AP at IoU level [0.85]: 0.206
     AP at IoU level [0.90]: 0.055
     AP at IoU level [0.95]: 0.001
Epoch: [5]  [  0/170]  eta: 0:01:03  lr: 0.000025  loss: 0.1010
(0.1010)  loss_classifier: 0.0321 (0.0321)  loss_box_reg: 0.0674
(0.0674)  loss_objectness: 0.0003 (0.0003)  loss_rpn_box_reg: 0.0013
(0.0013)  time: 0.3727  data: 0.0593  max mem: 2794
Epoch: [5]  [ 20/170]  eta: 0:00:53  lr: 0.000025  loss: 0.1438
(0.1488)  loss_classifier: 0.0414 (0.0437)  loss_box_reg: 0.0904
(0.0908)  loss_objectness: 0.0013 (0.0040)  loss_rpn_box_reg: 0.0024
(0.0103)  time: 0.3576  data: 0.0631  max mem: 2794
Epoch: [5]  [ 40/170]  eta: 0:00:46  lr: 0.000025  loss: 0.1490
(0.1590)  loss_classifier: 0.0439 (0.0456)  loss_box_reg: 0.1017
(0.0978)  loss_objectness: 0.0010 (0.0047)  loss_rpn_box_reg: 0.0032
(0.0109)  time: 0.3626  data: 0.0618  max mem: 2794
Epoch: [5]  [ 60/170]  eta: 0:00:39  lr: 0.000025  loss: 0.1295
(0.1487)  loss_classifier: 0.0365 (0.0431)  loss_box_reg: 0.0722
(0.0921)  loss_objectness: 0.0005 (0.0038)  loss_rpn_box_reg: 0.0019
(0.0097)  time: 0.3575  data: 0.0562  max mem: 2794
Epoch: [5]  [ 80/170]  eta: 0:00:32  lr: 0.000025  loss: 0.1244
(0.1461)  loss_classifier: 0.0373 (0.0422)  loss_box_reg: 0.0751
(0.0907)  loss_objectness: 0.0004 (0.0035)  loss_rpn_box_reg: 0.0033
(0.0097)  time: 0.3772  data: 0.0607  max mem: 2794
Epoch: [5]  [100/170]  eta: 0:00:25  lr: 0.000025  loss: 0.1151
(0.1442)  loss_classifier: 0.0351 (0.0417)  loss_box_reg: 0.0781
(0.0899)  loss_objectness: 0.0014 (0.0033)  loss_rpn_box_reg: 0.0030
(0.0093)  time: 0.3751  data: 0.0667  max mem: 2794
Epoch: [5]  [120/170]  eta: 0:00:18  lr: 0.000025  loss: 0.1620
(0.1469)  loss_classifier: 0.0380 (0.0421)  loss_box_reg: 0.1007
(0.0912)  loss_objectness: 0.0014 (0.0034)  loss_rpn_box_reg: 0.0059
(0.0103)  time: 0.3650  data: 0.0598  max mem: 2794
Epoch: [5]  [140/170]  eta: 0:00:10  lr: 0.000025  loss: 0.1413
(0.1453)  loss_classifier: 0.0416 (0.0415)  loss_box_reg: 0.0846
(0.0906)  loss_objectness: 0.0011 (0.0031)  loss_rpn_box_reg: 0.0037
```

```
(0.0100)  time: 0.3591  data: 0.0564  max mem: 2794
Epoch: [5]  [160/170]  eta: 0:00:03  lr: 0.000025  loss: 0.1331
(0.1453)  loss_classifier: 0.0421 (0.0415)  loss_box_reg: 0.0821
(0.0909)  loss_objectness: 0.0011 (0.0030)  loss_rpn_box_reg: 0.0038
(0.0098)  time: 0.3589  data: 0.0617  max mem: 2794
Epoch: [5]  [169/170]  eta: 0:00:00  lr: 0.000025  loss: 0.1167
(0.1443)  loss_classifier: 0.0351 (0.0412)  loss_box_reg: 0.0795
(0.0905)  loss_objectness: 0.0008 (0.0030)  loss_rpn_box_reg: 0.0027
(0.0097)  time: 0.3661  data: 0.0622  max mem: 2794
Epoch: [5] Total time: 0:01:01 (0.3645 s / it)
mAP:0.466
     AP at IoU level [0.50]: 0.708
     AP at IoU level [0.55]: 0.708
     AP at IoU level [0.60]: 0.681
     AP at IoU level [0.65]: 0.662
     AP at IoU level [0.70]: 0.640
     AP at IoU level [0.75]: 0.556
     AP at IoU level [0.80]: 0.417
     AP at IoU level [0.85]: 0.260
     AP at IoU level [0.90]: 0.026
     AP at IoU level [0.95]: 0.000
Epoch: [6]  [  0/170]  eta: 0:01:02  lr: 0.000013  loss: 0.0843
(0.0843)  loss_classifier: 0.0194 (0.0194)  loss_box_reg: 0.0589
(0.0589)  loss_objectness: 0.0034 (0.0034)  loss_rpn_box_reg: 0.0026
(0.0026)  time: 0.3689  data: 0.0560  max mem: 2794
Epoch: [6]  [ 20/170]  eta: 0:00:53  lr: 0.000013  loss: 0.1094
(0.1265)  loss_classifier: 0.0352 (0.0348)  loss_box_reg: 0.0653
(0.0788)  loss_objectness: 0.0009 (0.0036)  loss_rpn_box_reg: 0.0028
(0.0092)  time: 0.3589  data: 0.0652  max mem: 2794
Epoch: [6]  [ 40/170]  eta: 0:00:46  lr: 0.000013  loss: 0.1089
(0.1248)  loss_classifier: 0.0313 (0.0359)  loss_box_reg: 0.0672
(0.0756)  loss_objectness: 0.0005 (0.0041)  loss_rpn_box_reg: 0.0016
(0.0091)  time: 0.3590  data: 0.0612  max mem: 2794
Epoch: [6]  [ 60/170]  eta: 0:00:39  lr: 0.000013  loss: 0.1017
(0.1232)  loss_classifier: 0.0351 (0.0368)  loss_box_reg: 0.0616
(0.0751)  loss_objectness: 0.0006 (0.0033)  loss_rpn_box_reg: 0.0023
(0.0080)  time: 0.3597  data: 0.0622  max mem: 2794
Epoch: [6]  [ 80/170]  eta: 0:00:32  lr: 0.000013  loss: 0.0913
(0.1189)  loss_classifier: 0.0286 (0.0356)  loss_box_reg: 0.0616
(0.0735)  loss_objectness: 0.0003 (0.0027)  loss_rpn_box_reg: 0.0019
(0.0072)  time: 0.3677  data: 0.0606  max mem: 2794
Epoch: [6]  [100/170]  eta: 0:00:25  lr: 0.000013  loss: 0.1161
(0.1207)  loss_classifier: 0.0313 (0.0359)  loss_box_reg: 0.0726
(0.0747)  loss_objectness: 0.0009 (0.0028)  loss_rpn_box_reg: 0.0038
(0.0074)  time: 0.3681  data: 0.0658  max mem: 2794
Epoch: [6]  [120/170]  eta: 0:00:18  lr: 0.000013  loss: 0.1258
(0.1220)  loss_classifier: 0.0343 (0.0358)  loss_box_reg: 0.0771
(0.0758)  loss_objectness: 0.0009 (0.0029)  loss_rpn_box_reg: 0.0031
(0.0075)  time: 0.3541  data: 0.0599  max mem: 2794
Epoch: [6]  [140/170]  eta: 0:00:10  lr: 0.000013  loss: 0.0863
```

```
(0.1204)  loss_classifier: 0.0266 (0.0355)  loss_box_reg: 0.0585
(0.0748)  loss_objectness: 0.0003 (0.0028)  loss_rpn_box_reg: 0.0016
(0.0073)  time: 0.3559  data: 0.0570  max mem: 2794
Epoch: [6]  [160/170]  eta: 0:00:03  lr: 0.000013  loss: 0.1283
(0.1226)  loss_classifier: 0.0345 (0.0358)  loss_box_reg: 0.0864
(0.0758)  loss_objectness: 0.0017 (0.0029)  loss_rpn_box_reg: 0.0040
(0.0080)  time: 0.3624  data: 0.0589  max mem: 2794
Epoch: [6]  [169/170]  eta: 0:00:00  lr: 0.000013  loss: 0.1326
(0.1234)  loss_classifier: 0.0376 (0.0357)  loss_box_reg: 0.0864
(0.0766)  loss_objectness: 0.0011 (0.0029)  loss_rpn_box_reg: 0.0054
(0.0083)  time: 0.3681  data: 0.0604  max mem: 2794
Epoch: [6] Total time: 0:01:01 (0.3610 s / it)
mAP:0.472
     AP at IoU level [0.50]: 0.707
     AP at IoU level [0.55]: 0.699
     AP at IoU level [0.60]: 0.679
     AP at IoU level [0.65]: 0.661
     AP at IoU level [0.70]: 0.645
     AP at IoU level [0.75]: 0.569
     AP at IoU level [0.80]: 0.438
     AP at IoU level [0.85]: 0.266
     AP at IoU level [0.90]: 0.054
     AP at IoU level [0.95]: 0.001
Epoch: [7]  [  0/170]  eta: 0:01:03  lr: 0.000013  loss: 0.0580
(0.0580)  loss_classifier: 0.0123 (0.0123)  loss_box_reg: 0.0403
(0.0403)  loss_objectness: 0.0014 (0.0014)  loss_rpn_box_reg: 0.0040
(0.0040)  time: 0.3730  data: 0.0612  max mem: 2794
Epoch: [7]  [ 20/170]  eta: 0:00:54  lr: 0.000013  loss: 0.1260
(0.1335)  loss_classifier: 0.0392 (0.0399)  loss_box_reg: 0.0766
(0.0802)  loss_objectness: 0.0008 (0.0037)  loss_rpn_box_reg: 0.0036
(0.0096)  time: 0.3628  data: 0.0623  max mem: 2794
Epoch: [7]  [ 40/170]  eta: 0:00:46  lr: 0.000013  loss: 0.1007
(0.1231)  loss_classifier: 0.0310 (0.0364)  loss_box_reg: 0.0667
(0.0741)  loss_objectness: 0.0003 (0.0037)  loss_rpn_box_reg: 0.0015
(0.0089)  time: 0.3548  data: 0.0584  max mem: 2794
Epoch: [7]  [ 60/170]  eta: 0:00:39  lr: 0.000013  loss: 0.0962
(0.1178)  loss_classifier: 0.0272 (0.0352)  loss_box_reg: 0.0595
(0.0720)  loss_objectness: 0.0008 (0.0029)  loss_rpn_box_reg: 0.0033
(0.0077)  time: 0.3710  data: 0.0613  max mem: 2794
Epoch: [7]  [ 80/170]  eta: 0:00:32  lr: 0.000013  loss: 0.0842
(0.1133)  loss_classifier: 0.0277 (0.0341)  loss_box_reg: 0.0560
(0.0688)  loss_objectness: 0.0003 (0.0026)  loss_rpn_box_reg: 0.0024
(0.0078)  time: 0.3604  data: 0.0605  max mem: 2794
Epoch: [7]  [100/170]  eta: 0:00:25  lr: 0.000013  loss: 0.0738
(0.1078)  loss_classifier: 0.0224 (0.0325)  loss_box_reg: 0.0495
(0.0654)  loss_objectness: 0.0002 (0.0025)  loss_rpn_box_reg: 0.0012
(0.0075)  time: 0.3573  data: 0.0630  max mem: 2794
Epoch: [7]  [120/170]  eta: 0:00:18  lr: 0.000013  loss: 0.0805
(0.1078)  loss_classifier: 0.0290 (0.0321)  loss_box_reg: 0.0508
(0.0650)  loss_objectness: 0.0006 (0.0026)  loss_rpn_box_reg: 0.0023
```

```
(0.0081)  time: 0.3596  data: 0.0590  max mem: 2794
Epoch: [7]  [140/170]  eta: 0:00:10  lr: 0.000013  loss: 0.0982
(0.1079)  loss_classifier: 0.0294 (0.0324)  loss_box_reg: 0.0628
(0.0651)  loss_objectness: 0.0007 (0.0025)  loss_rpn_box_reg: 0.0028
(0.0079)  time: 0.3655  data: 0.0609  max mem: 2794
Epoch: [7]  [160/170]  eta: 0:00:03  lr: 0.000013  loss: 0.0855
(0.1084)  loss_classifier: 0.0284 (0.0326)  loss_box_reg: 0.0480
(0.0654)  loss_objectness: 0.0006 (0.0026)  loss_rpn_box_reg: 0.0016
(0.0079)  time: 0.3530  data: 0.0588  max mem: 2794
Epoch: [7]  [169/170]  eta: 0:00:00  lr: 0.000013  loss: 0.0926
(0.1076)  loss_classifier: 0.0289 (0.0324)  loss_box_reg: 0.0594
(0.0649)  loss_objectness: 0.0014 (0.0026)  loss_rpn_box_reg: 0.0039
(0.0077)  time: 0.3675  data: 0.0628  max mem: 2794
Epoch: [7] Total time: 0:01:01 (0.3620 s / it)
mAP:0.464
     AP at IoU level [0.50]: 0.717
     AP at IoU level [0.55]: 0.710
     AP at IoU level [0.60]: 0.688
     AP at IoU level [0.65]: 0.671
     AP at IoU level [0.70]: 0.632
     AP at IoU level [0.75]: 0.544
     AP at IoU level [0.80]: 0.376
     AP at IoU level [0.85]: 0.258
     AP at IoU level [0.90]: 0.038
     AP at IoU level [0.95]: 0.000
Epoch: [8]  [  0/170]  eta: 0:00:57  lr: 0.000006  loss: 0.0945
(0.0945)  loss_classifier: 0.0234 (0.0234)  loss_box_reg: 0.0614
(0.0614)  loss_objectness: 0.0010 (0.0010)  loss_rpn_box_reg: 0.0087
(0.0087)  time: 0.3396  data: 0.0516  max mem: 2794
Epoch: [8]  [ 20/170]  eta: 0:00:54  lr: 0.000006  loss: 0.0794
(0.1004)  loss_classifier: 0.0319 (0.0300)  loss_box_reg: 0.0472
(0.0567)  loss_objectness: 0.0006 (0.0044)  loss_rpn_box_reg: 0.0035
(0.0094)  time: 0.3649  data: 0.0588  max mem: 2794
Epoch: [8]  [ 40/170]  eta: 0:00:46  lr: 0.000006  loss: 0.0817
(0.0969)  loss_classifier: 0.0274 (0.0300)  loss_box_reg: 0.0477
(0.0558)  loss_objectness: 0.0002 (0.0039)  loss_rpn_box_reg: 0.0014
(0.0072)  time: 0.3453  data: 0.0618  max mem: 2794
Epoch: [8]  [ 60/170]  eta: 0:00:39  lr: 0.000006  loss: 0.0966
(0.1053)  loss_classifier: 0.0304 (0.0314)  loss_box_reg: 0.0616
(0.0616)  loss_objectness: 0.0003 (0.0034)  loss_rpn_box_reg: 0.0033
(0.0088)  time: 0.3681  data: 0.0591  max mem: 2794
Epoch: [8]  [ 80/170]  eta: 0:00:32  lr: 0.000006  loss: 0.0829
(0.1030)  loss_classifier: 0.0280 (0.0309)  loss_box_reg: 0.0500
(0.0609)  loss_objectness: 0.0003 (0.0032)  loss_rpn_box_reg: 0.0023
(0.0080)  time: 0.3675  data: 0.0640  max mem: 2794
Epoch: [8]  [100/170]  eta: 0:00:25  lr: 0.000006  loss: 0.0624
(0.0971)  loss_classifier: 0.0212 (0.0295)  loss_box_reg: 0.0381
(0.0576)  loss_objectness: 0.0006 (0.0028)  loss_rpn_box_reg: 0.0017
(0.0072)  time: 0.3708  data: 0.0624  max mem: 2794
Epoch: [8]  [120/170]  eta: 0:00:18  lr: 0.000006  loss: 0.0732
```

```
(0.0947)  loss_classifier: 0.0268 (0.0293)  loss_box_reg: 0.0396
(0.0559)  loss_objectness: 0.0003 (0.0025)  loss_rpn_box_reg: 0.0023
(0.0070)  time: 0.3544  data: 0.0595  max mem: 2794
Epoch: [8]  [140/170]  eta: 0:00:10  lr: 0.000006  loss: 0.0844
(0.0950)  loss_classifier: 0.0267 (0.0295)  loss_box_reg: 0.0484
(0.0559)  loss_objectness: 0.0004 (0.0025)  loss_rpn_box_reg: 0.0020
(0.0071)  time: 0.3534  data: 0.0599  max mem: 2794
Epoch: [8]  [160/170]  eta: 0:00:03  lr: 0.000006  loss: 0.0880
(0.0947)  loss_classifier: 0.0274 (0.0295)  loss_box_reg: 0.0532
(0.0561)  loss_objectness: 0.0002 (0.0023)  loss_rpn_box_reg: 0.0028
(0.0068)  time: 0.3597  data: 0.0577  max mem: 2794
Epoch: [8]  [169/170]  eta: 0:00:00  lr: 0.000006  loss: 0.0928
(0.0960)  loss_classifier: 0.0276 (0.0296)  loss_box_reg: 0.0547
(0.0569)  loss_objectness: 0.0010 (0.0024)  loss_rpn_box_reg: 0.0035
(0.0071)  time: 0.3558  data: 0.0588  max mem: 2794
Epoch: [8] Total time: 0:01:01 (0.3605 s / it)
mAP:0.462
     AP at IoU level [0.50]: 0.715
     AP at IoU level [0.55]: 0.698
     AP at IoU level [0.60]: 0.678
     AP at IoU level [0.65]: 0.662
     AP at IoU level [0.70]: 0.631
     AP at IoU level [0.75]: 0.549
     AP at IoU level [0.80]: 0.378
     AP at IoU level [0.85]: 0.261
     AP at IoU level [0.90]: 0.049
     AP at IoU level [0.95]: 0.001
Epoch: [9]  [  0/170]  eta: 0:01:02  lr: 0.000006  loss: 0.1462
(0.1462)  loss_classifier: 0.0343 (0.0343)  loss_box_reg: 0.1022
(0.1022)  loss_objectness: 0.0014 (0.0014)  loss_rpn_box_reg: 0.0083
(0.0083)  time: 0.3681  data: 0.0603  max mem: 2794
Epoch: [9]  [ 20/170]  eta: 0:00:53  lr: 0.000006  loss: 0.0819
(0.0990)  loss_classifier: 0.0294 (0.0332)  loss_box_reg: 0.0502
(0.0579)  loss_objectness: 0.0003 (0.0020)  loss_rpn_box_reg: 0.0016
(0.0059)  time: 0.3577  data: 0.0604  max mem: 2794
Epoch: [9]  [ 40/170]  eta: 0:00:46  lr: 0.000006  loss: 0.0554
(0.0850)  loss_classifier: 0.0205 (0.0287)  loss_box_reg: 0.0311
(0.0494)  loss_objectness: 0.0001 (0.0015)  loss_rpn_box_reg: 0.0006
(0.0054)  time: 0.3616  data: 0.0577  max mem: 2794
Epoch: [9]  [ 60/170]  eta: 0:00:39  lr: 0.000006  loss: 0.0893
(0.0863)  loss_classifier: 0.0258 (0.0290)  loss_box_reg: 0.0508
(0.0497)  loss_objectness: 0.0006 (0.0017)  loss_rpn_box_reg: 0.0056
(0.0059)  time: 0.3516  data: 0.0633  max mem: 2794
Epoch: [9]  [ 80/170]  eta: 0:00:32  lr: 0.000006  loss: 0.0631
(0.0871)  loss_classifier: 0.0207 (0.0284)  loss_box_reg: 0.0408
(0.0505)  loss_objectness: 0.0006 (0.0020)  loss_rpn_box_reg: 0.0023
(0.0063)  time: 0.3567  data: 0.0586  max mem: 2794
Epoch: [9]  [100/170]  eta: 0:00:25  lr: 0.000006  loss: 0.0671
(0.0891)  loss_classifier: 0.0250 (0.0287)  loss_box_reg: 0.0398
(0.0511)  loss_objectness: 0.0005 (0.0023)  loss_rpn_box_reg: 0.0012
```

```
(0.0070)  time: 0.3680  data: 0.0610  max mem: 2794
Epoch: [9]  [120/170]  eta: 0:00:18  lr: 0.000006  loss: 0.0619
(0.0858)  loss_classifier: 0.0222 (0.0276)  loss_box_reg: 0.0387
(0.0498)  loss_objectness: 0.0002 (0.0021)  loss_rpn_box_reg: 0.0011
(0.0063)  time: 0.3646  data: 0.0604  max mem: 2794
Epoch: [9]  [140/170]  eta: 0:00:10  lr: 0.000006  loss: 0.0866
(0.0877)  loss_classifier: 0.0272 (0.0284)  loss_box_reg: 0.0535
(0.0509)  loss_objectness: 0.0008 (0.0021)  loss_rpn_box_reg: 0.0030
(0.0064)  time: 0.3612  data: 0.0615  max mem: 2794
Epoch: [9]  [160/170]  eta: 0:00:03  lr: 0.000006  loss: 0.0665
(0.0887)  loss_classifier: 0.0296 (0.0286)  loss_box_reg: 0.0404
(0.0512)  loss_objectness: 0.0001 (0.0023)  loss_rpn_box_reg: 0.0018
(0.0067)  time: 0.3473  data: 0.0573  max mem: 2794
Epoch: [9]  [169/170]  eta: 0:00:00  lr: 0.000006  loss: 0.0628
(0.0898)  loss_classifier: 0.0275 (0.0288)  loss_box_reg: 0.0335
(0.0518)  loss_objectness: 0.0001 (0.0024)  loss_rpn_box_reg: 0.0013
(0.0067)  time: 0.3523  data: 0.0573  max mem: 2794
Epoch: [9] Total time: 0:01:00 (0.3585 s / it)
mAP:0.474
     AP at IoU level [0.50]: 0.715
     AP at IoU level [0.55]: 0.698
     AP at IoU level [0.60]: 0.677
     AP at IoU level [0.65]: 0.669
     AP at IoU level [0.70]: 0.631
     AP at IoU level [0.75]: 0.578
     AP at IoU level [0.80]: 0.452
     AP at IoU level [0.85]: 0.261
     AP at IoU level [0.90]: 0.061
     AP at IoU level [0.95]: 0.002
Epoch: [10]  [  0/170]  eta: 0:01:03  lr: 0.000003  loss: 0.0833
(0.0833)  loss_classifier: 0.0289 (0.0289)  loss_box_reg: 0.0525
(0.0525)  loss_objectness: 0.0001 (0.0001)  loss_rpn_box_reg: 0.0018
(0.0018)  time: 0.3711  data: 0.0585  max mem: 2794
Epoch: [10]  [ 20/170]  eta: 0:00:53  lr: 0.000003  loss: 0.0820
(0.0726)  loss_classifier: 0.0228 (0.0241)  loss_box_reg: 0.0478
(0.0437)  loss_objectness: 0.0005 (0.0011)  loss_rpn_box_reg: 0.0031
(0.0037)  time: 0.3570  data: 0.0581  max mem: 2794
Epoch: [10]  [ 40/170]  eta: 0:00:46  lr: 0.000003  loss: 0.0582
(0.0752)  loss_classifier: 0.0243 (0.0254)  loss_box_reg: 0.0303
(0.0432)  loss_objectness: 0.0001 (0.0018)  loss_rpn_box_reg: 0.0011
(0.0049)  time: 0.3636  data: 0.0614  max mem: 2794
Epoch: [10]  [ 60/170]  eta: 0:00:39  lr: 0.000003  loss: 0.0898
(0.0811)  loss_classifier: 0.0289 (0.0268)  loss_box_reg: 0.0524
(0.0462)  loss_objectness: 0.0007 (0.0020)  loss_rpn_box_reg: 0.0051
(0.0061)  time: 0.3648  data: 0.0603  max mem: 2794
Epoch: [10]  [ 80/170]  eta: 0:00:32  lr: 0.000003  loss: 0.0747
(0.0821)  loss_classifier: 0.0263 (0.0277)  loss_box_reg: 0.0466
(0.0467)  loss_objectness: 0.0003 (0.0018)  loss_rpn_box_reg: 0.0022
(0.0059)  time: 0.3589  data: 0.0623  max mem: 2794
Epoch: [10]  [100/170]  eta: 0:00:25  lr: 0.000003  loss: 0.0841
```

```
(0.0845)  loss_classifier: 0.0293 (0.0284)  loss_box_reg: 0.0511
(0.0477)  loss_objectness: 0.0008 (0.0019)  loss_rpn_box_reg: 0.0025
(0.0065)  time: 0.3758  data: 0.0691  max mem: 2794
Epoch: [10]  [120/170]  eta: 0:00:18  lr: 0.000003  loss: 0.0620
(0.0863)  loss_classifier: 0.0270 (0.0286)  loss_box_reg: 0.0354
(0.0488)  loss_objectness: 0.0002 (0.0022)  loss_rpn_box_reg: 0.0022
(0.0066)  time: 0.3605  data: 0.0585  max mem: 2794
Epoch: [10]  [140/170]  eta: 0:00:10  lr: 0.000003  loss: 0.0519
(0.0836)  loss_classifier: 0.0216 (0.0278)  loss_box_reg: 0.0263
(0.0475)  loss_objectness: 0.0002 (0.0021)  loss_rpn_box_reg: 0.0010
(0.0063)  time: 0.3464  data: 0.0578  max mem: 2794
Epoch: [10]  [160/170]  eta: 0:00:03  lr: 0.000003  loss: 0.0636
(0.0848)  loss_classifier: 0.0250 (0.0280)  loss_box_reg: 0.0392
(0.0480)  loss_objectness: 0.0005 (0.0022)  loss_rpn_box_reg: 0.0019
(0.0067)  time: 0.3644  data: 0.0638  max mem: 2794
Epoch: [10]  [169/170]  eta: 0:00:00  lr: 0.000003  loss: 0.0594
(0.0837)  loss_classifier: 0.0236 (0.0278)  loss_box_reg: 0.0333
(0.0473)  loss_objectness: 0.0003 (0.0021)  loss_rpn_box_reg: 0.0013
(0.0065)  time: 0.3542  data: 0.0605  max mem: 2794
Epoch: [10] Total time: 0:01:01 (0.3612 s / it)
mAP:0.459
     AP at IoU level [0.50]: 0.707
     AP at IoU level [0.55]: 0.698
     AP at IoU level [0.60]: 0.676
     AP at IoU level [0.65]: 0.651
     AP at IoU level [0.70]: 0.631
     AP at IoU level [0.75]: 0.543
     AP at IoU level [0.80]: 0.399
     AP at IoU level [0.85]: 0.242
     AP at IoU level [0.90]: 0.047
     AP at IoU level [0.95]: 0.000
Epoch: [11]  [  0/170]  eta: 0:01:10  lr: 0.000003  loss: 0.0280
(0.0280)  loss_classifier: 0.0128 (0.0128)  loss_box_reg: 0.0148
(0.0148)  loss_objectness: 0.0000 (0.0000)  loss_rpn_box_reg: 0.0004
(0.0004)  time: 0.4151  data: 0.0582  max mem: 2794
Epoch: [11]  [ 20/170]  eta: 0:00:56  lr: 0.000003  loss: 0.0641
(0.0688)  loss_classifier: 0.0221 (0.0244)  loss_box_reg: 0.0339
(0.0395)  loss_objectness: 0.0002 (0.0014)  loss_rpn_box_reg: 0.0020
(0.0036)  time: 0.3736  data: 0.0632  max mem: 2795
Epoch: [11]  [ 40/170]  eta: 0:00:47  lr: 0.000003  loss: 0.0525
(0.0681)  loss_classifier: 0.0218 (0.0244)  loss_box_reg: 0.0318
(0.0389)  loss_objectness: 0.0002 (0.0016)  loss_rpn_box_reg: 0.0012
(0.0031)  time: 0.3508  data: 0.0614  max mem: 2795
Epoch: [11]  [ 60/170]  eta: 0:00:39  lr: 0.000003  loss: 0.0757
(0.0736)  loss_classifier: 0.0231 (0.0251)  loss_box_reg: 0.0435
(0.0418)  loss_objectness: 0.0007 (0.0018)  loss_rpn_box_reg: 0.0021
(0.0049)  time: 0.3618  data: 0.0565  max mem: 2795
Epoch: [11]  [ 80/170]  eta: 0:00:32  lr: 0.000003  loss: 0.0645
(0.0728)  loss_classifier: 0.0253 (0.0248)  loss_box_reg: 0.0370
(0.0412)  loss_objectness: 0.0003 (0.0018)  loss_rpn_box_reg: 0.0017
```

```
(0.0049)  time: 0.3633  data: 0.0597  max mem: 2795
Epoch: [11]  [100/170]  eta: 0:00:25  lr: 0.000003  loss: 0.0882
(0.0788)  loss_classifier: 0.0285 (0.0263)  loss_box_reg: 0.0491
(0.0444)  loss_objectness: 0.0009 (0.0020)  loss_rpn_box_reg: 0.0044
(0.0060)  time: 0.3744  data: 0.0665  max mem: 2795
Epoch: [11]  [120/170]  eta: 0:00:18  lr: 0.000003  loss: 0.0608
(0.0776)  loss_classifier: 0.0210 (0.0261)  loss_box_reg: 0.0338
(0.0439)  loss_objectness: 0.0002 (0.0018)  loss_rpn_box_reg: 0.0012
(0.0058)  time: 0.3498  data: 0.0596  max mem: 2795
Epoch: [11]  [140/170]  eta: 0:00:10  lr: 0.000003  loss: 0.0481
(0.0769)  loss_classifier: 0.0210 (0.0259)  loss_box_reg: 0.0240
(0.0433)  loss_objectness: 0.0001 (0.0018)  loss_rpn_box_reg: 0.0010
(0.0059)  time: 0.3678  data: 0.0596  max mem: 2795
Epoch: [11]  [160/170]  eta: 0:00:03  lr: 0.000003  loss: 0.0725
(0.0788)  loss_classifier: 0.0275 (0.0265)  loss_box_reg: 0.0436
(0.0443)  loss_objectness: 0.0004 (0.0020)  loss_rpn_box_reg: 0.0024
(0.0060)  time: 0.3500  data: 0.0623  max mem: 2795
Epoch: [11]  [169/170]  eta: 0:00:00  lr: 0.000003  loss: 0.0823
(0.0801)  loss_classifier: 0.0295 (0.0267)  loss_box_reg: 0.0474
(0.0449)  loss_objectness: 0.0007 (0.0021)  loss_rpn_box_reg: 0.0030
(0.0063)  time: 0.3546  data: 0.0614  max mem: 2795
Epoch: [11] Total time: 0:01:01 (0.3617 s / it)
mAP:0.467
     AP at IoU level [0.50]: 0.715
     AP at IoU level [0.55]: 0.698
     AP at IoU level [0.60]: 0.676
     AP at IoU level [0.65]: 0.668
     AP at IoU level [0.70]: 0.629
     AP at IoU level [0.75]: 0.581
     AP at IoU level [0.80]: 0.420
     AP at IoU level [0.85]: 0.248
     AP at IoU level [0.90]: 0.039
     AP at IoU level [0.95]: 0.000
Epoch: [12]  [  0/170]  eta: 0:01:05  lr: 0.000002  loss: 0.0345
(0.0345)  loss_classifier: 0.0172 (0.0172)  loss_box_reg: 0.0170
(0.0170)  loss_objectness: 0.0000 (0.0000)  loss_rpn_box_reg: 0.0003
(0.0003)  time: 0.3872  data: 0.0527  max mem: 2795
Epoch: [12]  [ 20/170]  eta: 0:00:55  lr: 0.000002  loss: 0.0652
(0.0682)  loss_classifier: 0.0233 (0.0242)  loss_box_reg: 0.0377
(0.0367)  loss_objectness: 0.0003 (0.0013)  loss_rpn_box_reg: 0.0019
(0.0060)  time: 0.3686  data: 0.0652  max mem: 2795
Epoch: [12]  [ 40/170]  eta: 0:00:47  lr: 0.000002  loss: 0.0747
(0.0781)  loss_classifier: 0.0271 (0.0264)  loss_box_reg: 0.0470
(0.0435)  loss_objectness: 0.0008 (0.0014)  loss_rpn_box_reg: 0.0038
(0.0067)  time: 0.3540  data: 0.0617  max mem: 2795
Epoch: [12]  [ 60/170]  eta: 0:00:39  lr: 0.000002  loss: 0.0567
(0.0780)  loss_classifier: 0.0225 (0.0261)  loss_box_reg: 0.0322
(0.0435)  loss_objectness: 0.0005 (0.0019)  loss_rpn_box_reg: 0.0035
(0.0065)  time: 0.3653  data: 0.0565  max mem: 2795
Epoch: [12]  [ 80/170]  eta: 0:00:32  lr: 0.000002  loss: 0.0824
```

```
(0.0807)  loss_classifier: 0.0238 (0.0261)  loss_box_reg: 0.0470
(0.0448)  loss_objectness: 0.0011 (0.0026)  loss_rpn_box_reg: 0.0025
(0.0072)  time: 0.3554  data: 0.0614  max mem: 2795
Epoch: [12]  [100/170]  eta: 0:00:25  lr: 0.000002  loss: 0.0684
(0.0804)  loss_classifier: 0.0265 (0.0264)  loss_box_reg: 0.0399
(0.0445)  loss_objectness: 0.0006 (0.0026)  loss_rpn_box_reg: 0.0023
(0.0070)  time: 0.3663  data: 0.0683  max mem: 2795
Epoch: [12]  [120/170]  eta: 0:00:17  lr: 0.000002  loss: 0.0510
(0.0793)  loss_classifier: 0.0212 (0.0261)  loss_box_reg: 0.0272
(0.0439)  loss_objectness: 0.0001 (0.0025)  loss_rpn_box_reg: 0.0008
(0.0069)  time: 0.3489  data: 0.0570  max mem: 2795
Epoch: [12]  [140/170]  eta: 0:00:10  lr: 0.000002  loss: 0.0612
(0.0781)  loss_classifier: 0.0248 (0.0261)  loss_box_reg: 0.0319
(0.0432)  loss_objectness: 0.0004 (0.0023)  loss_rpn_box_reg: 0.0020
(0.0065)  time: 0.3657  data: 0.0614  max mem: 2795
Epoch: [12]  [160/170]  eta: 0:00:03  lr: 0.000002  loss: 0.0562
(0.0766)  loss_classifier: 0.0231 (0.0259)  loss_box_reg: 0.0311
(0.0423)  loss_objectness: 0.0001 (0.0022)  loss_rpn_box_reg: 0.0011
(0.0062)  time: 0.3522  data: 0.0588  max mem: 2795
Epoch: [12]  [169/170]  eta: 0:00:00  lr: 0.000002  loss: 0.0582
(0.0766)  loss_classifier: 0.0201 (0.0258)  loss_box_reg: 0.0325
(0.0423)  loss_objectness: 0.0003 (0.0023)  loss_rpn_box_reg: 0.0014
(0.0062)  time: 0.3541  data: 0.0634  max mem: 2795
Epoch: [12] Total time: 0:01:01 (0.3598 s / it)
mAP:0.466
     AP at IoU level [0.50]: 0.715
     AP at IoU level [0.55]: 0.698
     AP at IoU level [0.60]: 0.677
     AP at IoU level [0.65]: 0.669
     AP at IoU level [0.70]: 0.631
     AP at IoU level [0.75]: 0.574
     AP at IoU level [0.80]: 0.402
     AP at IoU level [0.85]: 0.246
     AP at IoU level [0.90]: 0.048
     AP at IoU level [0.95]: 0.000
Epoch: [13]  [  0/170]  eta: 0:01:05  lr: 0.000002  loss: 0.0385
(0.0385)  loss_classifier: 0.0164 (0.0164)  loss_box_reg: 0.0218
(0.0218)  loss_objectness: 0.0000 (0.0000)  loss_rpn_box_reg: 0.0002
(0.0002)  time: 0.3866  data: 0.0742  max mem: 2795
Epoch: [13]  [ 20/170]  eta: 0:00:55  lr: 0.000002  loss: 0.0547
(0.0756)  loss_classifier: 0.0209 (0.0247)  loss_box_reg: 0.0321
(0.0409)  loss_objectness: 0.0003 (0.0021)  loss_rpn_box_reg: 0.0021
(0.0079)  time: 0.3715  data: 0.0614  max mem: 2795
Epoch: [13]  [ 40/170]  eta: 0:00:48  lr: 0.000002  loss: 0.0516
(0.0726)  loss_classifier: 0.0184 (0.0238)  loss_box_reg: 0.0282
(0.0402)  loss_objectness: 0.0004 (0.0021)  loss_rpn_box_reg: 0.0017
(0.0064)  time: 0.3687  data: 0.0585  max mem: 2795
Epoch: [13]  [ 60/170]  eta: 0:00:40  lr: 0.000002  loss: 0.0641
(0.0739)  loss_classifier: 0.0250 (0.0246)  loss_box_reg: 0.0319
(0.0404)  loss_objectness: 0.0007 (0.0020)  loss_rpn_box_reg: 0.0017
```

```
(0.0068)  time: 0.3598  data: 0.0625  max mem: 2795
Epoch: [13]  [ 80/170]  eta: 0:00:32  lr: 0.000002  loss: 0.0693
(0.0733)  loss_classifier: 0.0265 (0.0248)  loss_box_reg: 0.0366
(0.0402)  loss_objectness: 0.0001 (0.0018)  loss_rpn_box_reg: 0.0017
(0.0065)  time: 0.3612  data: 0.0590  max mem: 2795
Epoch: [13]  [100/170]  eta: 0:00:25  lr: 0.000002  loss: 0.0726
(0.0780)  loss_classifier: 0.0259 (0.0264)  loss_box_reg: 0.0405
(0.0428)  loss_objectness: 0.0008 (0.0019)  loss_rpn_box_reg: 0.0051
(0.0069)  time: 0.3682  data: 0.0643  max mem: 2795
Epoch: [13]  [120/170]  eta: 0:00:18  lr: 0.000002  loss: 0.0619
(0.0763)  loss_classifier: 0.0202 (0.0258)  loss_box_reg: 0.0364
(0.0423)  loss_objectness: 0.0003 (0.0019)  loss_rpn_box_reg: 0.0015
(0.0064)  time: 0.3452  data: 0.0639  max mem: 2795
Epoch: [13]  [140/170]  eta: 0:00:10  lr: 0.000002  loss: 0.0593
(0.0758)  loss_classifier: 0.0218 (0.0256)  loss_box_reg: 0.0324
(0.0420)  loss_objectness: 0.0004 (0.0019)  loss_rpn_box_reg: 0.0026
(0.0063)  time: 0.3539  data: 0.0600  max mem: 2795
Epoch: [13]  [160/170]  eta: 0:00:03  lr: 0.000002  loss: 0.0761
(0.0761)  loss_classifier: 0.0281 (0.0259)  loss_box_reg: 0.0395
(0.0421)  loss_objectness: 0.0003 (0.0018)  loss_rpn_box_reg: 0.0024
(0.0063)  time: 0.3459  data: 0.0607  max mem: 2795
Epoch: [13]  [169/170]  eta: 0:00:00  lr: 0.000002  loss: 0.0628
(0.0751)  loss_classifier: 0.0251 (0.0257)  loss_box_reg: 0.0388
(0.0416)  loss_objectness: 0.0005 (0.0018)  loss_rpn_box_reg: 0.0025
(0.0061)  time: 0.3487  data: 0.0625  max mem: 2795
Epoch: [13] Total time: 0:01:01 (0.3594 s / it)
mAP:0.466
     AP at IoU level [0.50]: 0.715
     AP at IoU level [0.55]: 0.698
     AP at IoU level [0.60]: 0.677
     AP at IoU level [0.65]: 0.669
     AP at IoU level [0.70]: 0.631
     AP at IoU level [0.75]: 0.575
     AP at IoU level [0.80]: 0.403
     AP at IoU level [0.85]: 0.253
     AP at IoU level [0.90]: 0.036
     AP at IoU level [0.95]: 0.000
Epoch: [14]  [  0/170]  eta: 0:01:05  lr: 0.000001  loss: 0.0389
(0.0389)  loss_classifier: 0.0208 (0.0208)  loss_box_reg: 0.0177
(0.0177)  loss_objectness: 0.0000 (0.0000)  loss_rpn_box_reg: 0.0003
(0.0003)  time: 0.3834  data: 0.0489  max mem: 2795
Epoch: [14]  [ 20/170]  eta: 0:00:54  lr: 0.000001  loss: 0.0697
(0.0788)  loss_classifier: 0.0206 (0.0262)  loss_box_reg: 0.0374
(0.0409)  loss_objectness: 0.0003 (0.0029)  loss_rpn_box_reg: 0.0014
(0.0088)  time: 0.3590  data: 0.0658  max mem: 2795
Epoch: [14]  [ 40/170]  eta: 0:00:47  lr: 0.000001  loss: 0.0545
(0.0739)  loss_classifier: 0.0227 (0.0253)  loss_box_reg: 0.0285
(0.0400)  loss_objectness: 0.0003 (0.0022)  loss_rpn_box_reg: 0.0010
(0.0064)  time: 0.3638  data: 0.0617  max mem: 2795
Epoch: [14]  [ 60/170]  eta: 0:00:39  lr: 0.000001  loss: 0.0524
```

```
(0.0737)  loss_classifier: 0.0220 (0.0250)  loss_box_reg: 0.0349
(0.0397)  loss_objectness: 0.0005 (0.0022)  loss_rpn_box_reg: 0.0022
(0.0068)  time: 0.3637  data: 0.0567  max mem: 2795
Epoch: [14]  [ 80/170]  eta: 0:00:32  lr: 0.000001  loss: 0.0663
(0.0756)  loss_classifier: 0.0257 (0.0256)  loss_box_reg: 0.0353
(0.0409)  loss_objectness: 0.0004 (0.0025)  loss_rpn_box_reg: 0.0022
(0.0067)  time: 0.3593  data: 0.0626  max mem: 2795
Epoch: [14]  [100/170]  eta: 0:00:25  lr: 0.000001  loss: 0.0641
(0.0763)  loss_classifier: 0.0270 (0.0258)  loss_box_reg: 0.0317
(0.0414)  loss_objectness: 0.0002 (0.0025)  loss_rpn_box_reg: 0.0018
(0.0066)  time: 0.3714  data: 0.0657  max mem: 2795
Epoch: [14]  [120/170]  eta: 0:00:18  lr: 0.000001  loss: 0.0582
(0.0764)  loss_classifier: 0.0243 (0.0256)  loss_box_reg: 0.0298
(0.0416)  loss_objectness: 0.0007 (0.0024)  loss_rpn_box_reg: 0.0022
(0.0067)  time: 0.3630  data: 0.0616  max mem: 2795
Epoch: [14]  [140/170]  eta: 0:00:10  lr: 0.000001  loss: 0.0576
(0.0745)  loss_classifier: 0.0214 (0.0252)  loss_box_reg: 0.0343
(0.0406)  loss_objectness: 0.0002 (0.0023)  loss_rpn_box_reg: 0.0021
(0.0063)  time: 0.3679  data: 0.0593  max mem: 2795
Epoch: [14]  [160/170]  eta: 0:00:03  lr: 0.000001  loss: 0.0679
(0.0744)  loss_classifier: 0.0236 (0.0254)  loss_box_reg: 0.0347
(0.0408)  loss_objectness: 0.0001 (0.0022)  loss_rpn_box_reg: 0.0012
(0.0060)  time: 0.3686  data: 0.0625  max mem: 2795
Epoch: [14]  [169/170]  eta: 0:00:00  lr: 0.000001  loss: 0.0546
(0.0737)  loss_classifier: 0.0243 (0.0253)  loss_box_reg: 0.0318
(0.0404)  loss_objectness: 0.0002 (0.0021)  loss_rpn_box_reg: 0.0012
(0.0059)  time: 0.3631  data: 0.0601  max mem: 2795
Epoch: [14] Total time: 0:01:01 (0.3642 s / it)
mAP:0.469
      AP at IoU level [0.50]: 0.715
      AP at IoU level [0.55]: 0.698
      AP at IoU level [0.60]: 0.677
      AP at IoU level [0.65]: 0.669
      AP at IoU level [0.70]: 0.632
      AP at IoU level [0.75]: 0.575
      AP at IoU level [0.80]: 0.423
      AP at IoU level [0.85]: 0.260
      AP at IoU level [0.90]: 0.041
      AP at IoU level [0.95]: 0.000
Epoch: [15]  [  0/170]  eta: 0:01:01  lr: 0.000001  loss: 0.0404
(0.0404)  loss_classifier: 0.0167 (0.0167)  loss_box_reg: 0.0226
(0.0226)  loss_objectness: 0.0001 (0.0001)  loss_rpn_box_reg: 0.0010
(0.0010)  time: 0.3606  data: 0.0520  max mem: 2795
Epoch: [15]  [ 20/170]  eta: 0:00:53  lr: 0.000001  loss: 0.0383
(0.0671)  loss_classifier: 0.0174 (0.0226)  loss_box_reg: 0.0220
(0.0373)  loss_objectness: 0.0001 (0.0026)  loss_rpn_box_reg: 0.0008
(0.0047)  time: 0.3559  data: 0.0603  max mem: 2795
Epoch: [15]  [ 40/170]  eta: 0:00:46  lr: 0.000001  loss: 0.0706
(0.0782)  loss_classifier: 0.0221 (0.0255)  loss_box_reg: 0.0349
(0.0429)  loss_objectness: 0.0007 (0.0028)  loss_rpn_box_reg: 0.0014
```

```
(0.0070)  time: 0.3663  data: 0.0646  max mem: 2795
Epoch: [15]  [ 60/170]  eta: 0:00:39  lr: 0.000001  loss: 0.0605
(0.0749)  loss_classifier: 0.0254 (0.0256)  loss_box_reg: 0.0292
(0.0411)  loss_objectness: 0.0003 (0.0022)  loss_rpn_box_reg: 0.0014
(0.0060)  time: 0.3611  data: 0.0598  max mem: 2795
Epoch: [15]  [ 80/170]  eta: 0:00:32  lr: 0.000001  loss: 0.0648
(0.0739)  loss_classifier: 0.0224 (0.0254)  loss_box_reg: 0.0343
(0.0404)  loss_objectness: 0.0003 (0.0021)  loss_rpn_box_reg: 0.0023
(0.0060)  time: 0.3617  data: 0.0601  max mem: 2795
Epoch: [15]  [100/170]  eta: 0:00:25  lr: 0.000001  loss: 0.0710
(0.0733)  loss_classifier: 0.0234 (0.0252)  loss_box_reg: 0.0382
(0.0403)  loss_objectness: 0.0005 (0.0020)  loss_rpn_box_reg: 0.0048
(0.0058)  time: 0.3719  data: 0.0577  max mem: 2795
Epoch: [15]  [120/170]  eta: 0:00:18  lr: 0.000001  loss: 0.0666
(0.0744)  loss_classifier: 0.0256 (0.0256)  loss_box_reg: 0.0358
(0.0409)  loss_objectness: 0.0005 (0.0020)  loss_rpn_box_reg: 0.0016
(0.0059)  time: 0.3721  data: 0.0706  max mem: 2795
Epoch: [15]  [140/170]  eta: 0:00:10  lr: 0.000001  loss: 0.0715
(0.0768)  loss_classifier: 0.0240 (0.0260)  loss_box_reg: 0.0371
(0.0419)  loss_objectness: 0.0008 (0.0024)  loss_rpn_box_reg: 0.0030
(0.0066)  time: 0.3682  data: 0.0632  max mem: 2795
Epoch: [15]  [160/170]  eta: 0:00:03  lr: 0.000001  loss: 0.0545
(0.0747)  loss_classifier: 0.0203 (0.0257)  loss_box_reg: 0.0287
(0.0407)  loss_objectness: 0.0002 (0.0022)  loss_rpn_box_reg: 0.0011
(0.0062)  time: 0.3641  data: 0.0647  max mem: 2795
Epoch: [15]  [169/170]  eta: 0:00:00  lr: 0.000001  loss: 0.0639
(0.0738)  loss_classifier: 0.0227 (0.0254)  loss_box_reg: 0.0327
(0.0401)  loss_objectness: 0.0002 (0.0021)  loss_rpn_box_reg: 0.0020
(0.0061)  time: 0.3647  data: 0.0611  max mem: 2795
Epoch: [15] Total time: 0:01:02 (0.3651 s / it)
mAP:0.465
     AP at IoU level [0.50]: 0.714
     AP at IoU level [0.55]: 0.697
     AP at IoU level [0.60]: 0.676
     AP at IoU level [0.65]: 0.668
     AP at IoU level [0.70]: 0.631
     AP at IoU level [0.75]: 0.574
     AP at IoU level [0.80]: 0.402
     AP at IoU level [0.85]: 0.243
     AP at IoU level [0.90]: 0.046
     AP at IoU level [0.95]: 0.000
Epoch: [16]  [  0/170]  eta: 0:01:05  lr: 0.000000  loss: 0.0529
(0.0529)  loss_classifier: 0.0243 (0.0243)  loss_box_reg: 0.0276
(0.0276)  loss_objectness: 0.0001 (0.0001)  loss_rpn_box_reg: 0.0010
(0.0010)  time: 0.3842  data: 0.0983  max mem: 2795
Epoch: [16]  [ 20/170]  eta: 0:00:55  lr: 0.000000  loss: 0.0448
(0.0585)  loss_classifier: 0.0173 (0.0217)  loss_box_reg: 0.0258
(0.0317)  loss_objectness: 0.0002 (0.0009)  loss_rpn_box_reg: 0.0010
(0.0042)  time: 0.3703  data: 0.0588  max mem: 2795
Epoch: [16]  [ 40/170]  eta: 0:00:47  lr: 0.000000  loss: 0.0896
```

```
(0.0801)  loss_classifier: 0.0294 (0.0267)  loss_box_reg: 0.0441
(0.0430)  loss_objectness: 0.0011 (0.0029)  loss_rpn_box_reg: 0.0038
(0.0075)  time: 0.3576  data: 0.0603  max mem: 2795
Epoch: [16]  [ 60/170]  eta: 0:00:39  lr: 0.000000  loss: 0.0672
(0.0772)  loss_classifier: 0.0244 (0.0260)  loss_box_reg: 0.0336
(0.0417)  loss_objectness: 0.0010 (0.0025)  loss_rpn_box_reg: 0.0029
(0.0070)  time: 0.3603  data: 0.0586  max mem: 2795
Epoch: [16]  [ 80/170]  eta: 0:00:32  lr: 0.000000  loss: 0.0510
(0.0739)  loss_classifier: 0.0177 (0.0254)  loss_box_reg: 0.0293
(0.0398)  loss_objectness: 0.0002 (0.0021)  loss_rpn_box_reg: 0.0013
(0.0066)  time: 0.3655  data: 0.0619  max mem: 2795
Epoch: [16]  [100/170]  eta: 0:00:25  lr: 0.000000  loss: 0.0602
(0.0745)  loss_classifier: 0.0214 (0.0255)  loss_box_reg: 0.0332
(0.0402)  loss_objectness: 0.0014 (0.0020)  loss_rpn_box_reg: 0.0030
(0.0068)  time: 0.3708  data: 0.0632  max mem: 2795
Epoch: [16]  [120/170]  eta: 0:00:18  lr: 0.000000  loss: 0.0677
(0.0739)  loss_classifier: 0.0212 (0.0252)  loss_box_reg: 0.0369
(0.0403)  loss_objectness: 0.0002 (0.0019)  loss_rpn_box_reg: 0.0025
(0.0064)  time: 0.3688  data: 0.0653  max mem: 2795
Epoch: [16]  [140/170]  eta: 0:00:10  lr: 0.000000  loss: 0.0547
(0.0725)  loss_classifier: 0.0217 (0.0249)  loss_box_reg: 0.0302
(0.0395)  loss_objectness: 0.0002 (0.0020)  loss_rpn_box_reg: 0.0013
(0.0061)  time: 0.3613  data: 0.0608  max mem: 2795
Epoch: [16]  [160/170]  eta: 0:00:03  lr: 0.000000  loss: 0.0603
(0.0727)  loss_classifier: 0.0234 (0.0251)  loss_box_reg: 0.0361
(0.0395)  loss_objectness: 0.0002 (0.0019)  loss_rpn_box_reg: 0.0020
(0.0061)  time: 0.3561  data: 0.0600  max mem: 2795
Epoch: [16]  [169/170]  eta: 0:00:00  lr: 0.000000  loss: 0.0551
(0.0725)  loss_classifier: 0.0234 (0.0250)  loss_box_reg: 0.0315
(0.0395)  loss_objectness: 0.0004 (0.0019)  loss_rpn_box_reg: 0.0022
(0.0061)  time: 0.3587  data: 0.0623  max mem: 2795
Epoch: [16] Total time: 0:01:01 (0.3639 s / it)
mAP:0.467
     AP at IoU level [0.50]: 0.714
     AP at IoU level [0.55]: 0.697
     AP at IoU level [0.60]: 0.676
     AP at IoU level [0.65]: 0.668
     AP at IoU level [0.70]: 0.631
     AP at IoU level [0.75]: 0.574
     AP at IoU level [0.80]: 0.422
     AP at IoU level [0.85]: 0.250
     AP at IoU level [0.90]: 0.040
     AP at IoU level [0.95]: 0.000
Epoch: [17]  [  0/170]  eta: 0:01:04  lr: 0.000000  loss: 0.1328
(0.1328)  loss_classifier: 0.0380 (0.0380)  loss_box_reg: 0.0741
(0.0741)  loss_objectness: 0.0057 (0.0057)  loss_rpn_box_reg: 0.0150
(0.0150)  time: 0.3776  data: 0.0643  max mem: 2795
Epoch: [17]  [ 20/170]  eta: 0:00:55  lr: 0.000000  loss: 0.0692
(0.0774)  loss_classifier: 0.0247 (0.0251)  loss_box_reg: 0.0408
(0.0432)  loss_objectness: 0.0006 (0.0023)  loss_rpn_box_reg: 0.0039
```

```
(0.0068)  time: 0.3721  data: 0.0653  max mem: 2795
Epoch: [17]  [ 40/170]  eta: 0:00:47  lr: 0.000000  loss: 0.0638
(0.0821)  loss_classifier: 0.0257 (0.0266)  loss_box_reg: 0.0300
(0.0453)  loss_objectness: 0.0011 (0.0026)  loss_rpn_box_reg: 0.0027
(0.0077)  time: 0.3589  data: 0.0649  max mem: 2795
Epoch: [17]  [ 60/170]  eta: 0:00:40  lr: 0.000000  loss: 0.0686
(0.0797)  loss_classifier: 0.0212 (0.0264)  loss_box_reg: 0.0367
(0.0436)  loss_objectness: 0.0009 (0.0025)  loss_rpn_box_reg: 0.0021
(0.0072)  time: 0.3622  data: 0.0571  max mem: 2795
Epoch: [17]  [ 80/170]  eta: 0:00:32  lr: 0.000000  loss: 0.0609
(0.0795)  loss_classifier: 0.0242 (0.0267)  loss_box_reg: 0.0335
(0.0433)  loss_objectness: 0.0007 (0.0026)  loss_rpn_box_reg: 0.0018
(0.0068)  time: 0.3687  data: 0.0641  max mem: 2795
Epoch: [17]  [100/170]  eta: 0:00:25  lr: 0.000000  loss: 0.0459
(0.0754)  loss_classifier: 0.0207 (0.0258)  loss_box_reg: 0.0224
(0.0407)  loss_objectness: 0.0001 (0.0023)  loss_rpn_box_reg: 0.0008
(0.0067)  time: 0.3521  data: 0.0597  max mem: 2795
Epoch: [17]  [120/170]  eta: 0:00:18  lr: 0.000000  loss: 0.0442
(0.0723)  loss_classifier: 0.0202 (0.0250)  loss_box_reg: 0.0207
(0.0391)  loss_objectness: 0.0001 (0.0021)  loss_rpn_box_reg: 0.0009
(0.0061)  time: 0.3604  data: 0.0587  max mem: 2795
Epoch: [17]  [140/170]  eta: 0:00:10  lr: 0.000000  loss: 0.0499
(0.0700)  loss_classifier: 0.0173 (0.0245)  loss_box_reg: 0.0283
(0.0380)  loss_objectness: 0.0001 (0.0019)  loss_rpn_box_reg: 0.0018
(0.0056)  time: 0.3609  data: 0.0568  max mem: 2795
Epoch: [17]  [160/170]  eta: 0:00:03  lr: 0.000000  loss: 0.0739
(0.0717)  loss_classifier: 0.0252 (0.0248)  loss_box_reg: 0.0441
(0.0390)  loss_objectness: 0.0006 (0.0020)  loss_rpn_box_reg: 0.0024
(0.0060)  time: 0.3563  data: 0.0614  max mem: 2795
Epoch: [17]  [169/170]  eta: 0:00:00  lr: 0.000000  loss: 0.0739
(0.0719)  loss_classifier: 0.0252 (0.0248)  loss_box_reg: 0.0407
(0.0392)  loss_objectness: 0.0006 (0.0020)  loss_rpn_box_reg: 0.0024
(0.0060)  time: 0.3572  data: 0.0606  max mem: 2795
Epoch: [17] Total time: 0:01:01 (0.3611 s / it)
mAP:0.464
     AP at IoU level [0.50]: 0.707
     AP at IoU level [0.55]: 0.697
     AP at IoU level [0.60]: 0.677
     AP at IoU level [0.65]: 0.669
     AP at IoU level [0.70]: 0.632
     AP at IoU level [0.75]: 0.575
     AP at IoU level [0.80]: 0.402
     AP at IoU level [0.85]: 0.244
     AP at IoU level [0.90]: 0.040
     AP at IoU level [0.95]: 0.000
Epoch: [18]  [  0/170]  eta: 0:00:55  lr: 0.000000  loss: 0.0357
(0.0357)  loss_classifier: 0.0176 (0.0176)  loss_box_reg: 0.0176
(0.0176)  loss_objectness: 0.0000 (0.0000)  loss_rpn_box_reg: 0.0004
(0.0004)  time: 0.3290  data: 0.0692  max mem: 2795
Epoch: [18]  [ 20/170]  eta: 0:00:54  lr: 0.000000  loss: 0.0599
```

```
(0.0769)  loss_classifier: 0.0234 (0.0239)  loss_box_reg: 0.0361
(0.0439)  loss_objectness: 0.0004 (0.0016)  loss_rpn_box_reg: 0.0022
(0.0075)  time: 0.3619  data: 0.0612  max mem: 2795
Epoch: [18]  [ 40/170]  eta: 0:00:47  lr: 0.000000  loss: 0.0690
(0.0832)  loss_classifier: 0.0217 (0.0256)  loss_box_reg: 0.0399
(0.0475)  loss_objectness: 0.0005 (0.0026)  loss_rpn_box_reg: 0.0022
(0.0074)  time: 0.3691  data: 0.0630  max mem: 2795
Epoch: [18]  [ 60/170]  eta: 0:00:39  lr: 0.000000  loss: 0.0560
(0.0752)  loss_classifier: 0.0228 (0.0244)  loss_box_reg: 0.0276
(0.0425)  loss_objectness: 0.0002 (0.0022)  loss_rpn_box_reg: 0.0014
(0.0061)  time: 0.3593  data: 0.0599  max mem: 2795
Epoch: [18]  [ 80/170]  eta: 0:00:32  lr: 0.000000  loss: 0.0458
(0.0700)  loss_classifier: 0.0199 (0.0239)  loss_box_reg: 0.0248
(0.0389)  loss_objectness: 0.0001 (0.0020)  loss_rpn_box_reg: 0.0008
(0.0052)  time: 0.3637  data: 0.0594  max mem: 2795
Epoch: [18]  [100/170]  eta: 0:00:25  lr: 0.000000  loss: 0.0484
(0.0689)  loss_classifier: 0.0209 (0.0239)  loss_box_reg: 0.0266
(0.0375)  loss_objectness: 0.0002 (0.0018)  loss_rpn_box_reg: 0.0012
(0.0058)  time: 0.3656  data: 0.0555  max mem: 2795
Epoch: [18]  [120/170]  eta: 0:00:18  lr: 0.000000  loss: 0.0553
(0.0694)  loss_classifier: 0.0200 (0.0239)  loss_box_reg: 0.0341
(0.0380)  loss_objectness: 0.0003 (0.0019)  loss_rpn_box_reg: 0.0030
(0.0056)  time: 0.3736  data: 0.0637  max mem: 2795
Epoch: [18]  [140/170]  eta: 0:00:10  lr: 0.000000  loss: 0.0593
(0.0691)  loss_classifier: 0.0246 (0.0243)  loss_box_reg: 0.0345
(0.0378)  loss_objectness: 0.0002 (0.0017)  loss_rpn_box_reg: 0.0018
(0.0052)  time: 0.3643  data: 0.0627  max mem: 2795
Epoch: [18]  [160/170]  eta: 0:00:03  lr: 0.000000  loss: 0.0658
(0.0713)  loss_classifier: 0.0254 (0.0247)  loss_box_reg: 0.0314
(0.0390)  loss_objectness: 0.0009 (0.0018)  loss_rpn_box_reg: 0.0026
(0.0058)  time: 0.3610  data: 0.0598  max mem: 2795
Epoch: [18]  [169/170]  eta: 0:00:00  lr: 0.000000  loss: 0.0602
(0.0712)  loss_classifier: 0.0248 (0.0247)  loss_box_reg: 0.0298
(0.0388)  loss_objectness: 0.0001 (0.0018)  loss_rpn_box_reg: 0.0016
(0.0059)  time: 0.3602  data: 0.0611  max mem: 2795
Epoch: [18] Total time: 0:01:01 (0.3644 s / it)
mAP:0.466
     AP at IoU level [0.50]: 0.707
     AP at IoU level [0.55]: 0.698
     AP at IoU level [0.60]: 0.676
     AP at IoU level [0.65]: 0.668
     AP at IoU level [0.70]: 0.631
     AP at IoU level [0.75]: 0.574
     AP at IoU level [0.80]: 0.420
     AP at IoU level [0.85]: 0.245
     AP at IoU level [0.90]: 0.040
     AP at IoU level [0.95]: 0.000
Epoch: [19]  [  0/170]  eta: 0:01:03  lr: 0.000000  loss: 0.0642
(0.0642)  loss_classifier: 0.0244 (0.0244)  loss_box_reg: 0.0336
(0.0336)  loss_objectness: 0.0022 (0.0022)  loss_rpn_box_reg: 0.0040
```

```
(0.0040)  time: 0.3710  data: 0.0608  max mem: 2795
Epoch: [19]  [ 20/170]  eta: 0:00:53  lr: 0.000000  loss: 0.0741
(0.0830)  loss_classifier: 0.0262 (0.0284)  loss_box_reg: 0.0404
(0.0445)  loss_objectness: 0.0006 (0.0034)  loss_rpn_box_reg: 0.0031
(0.0067)  time: 0.3566  data: 0.0596  max mem: 2795
Epoch: [19]  [ 40/170]  eta: 0:00:47  lr: 0.000000  loss: 0.0513
(0.0805)  loss_classifier: 0.0202 (0.0275)  loss_box_reg: 0.0290
(0.0439)  loss_objectness: 0.0002 (0.0029)  loss_rpn_box_reg: 0.0009
(0.0063)  time: 0.3673  data: 0.0636  max mem: 2795
Epoch: [19]  [ 60/170]  eta: 0:00:39  lr: 0.000000  loss: 0.0682
(0.0789)  loss_classifier: 0.0240 (0.0271)  loss_box_reg: 0.0399
(0.0429)  loss_objectness: 0.0010 (0.0026)  loss_rpn_box_reg: 0.0020
(0.0064)  time: 0.3660  data: 0.0651  max mem: 2795
Epoch: [19]  [ 80/170]  eta: 0:00:32  lr: 0.000000  loss: 0.0649
(0.0775)  loss_classifier: 0.0239 (0.0265)  loss_box_reg: 0.0369
(0.0420)  loss_objectness: 0.0005 (0.0025)  loss_rpn_box_reg: 0.0014
(0.0066)  time: 0.3610  data: 0.0611  max mem: 2795
Epoch: [19]  [100/170]  eta: 0:00:25  lr: 0.000000  loss: 0.0563
(0.0779)  loss_classifier: 0.0235 (0.0266)  loss_box_reg: 0.0307
(0.0419)  loss_objectness: 0.0003 (0.0025)  loss_rpn_box_reg: 0.0022
(0.0069)  time: 0.3684  data: 0.0581  max mem: 2795
Epoch: [19]  [120/170]  eta: 0:00:18  lr: 0.000000  loss: 0.0449
(0.0758)  loss_classifier: 0.0210 (0.0259)  loss_box_reg: 0.0270
(0.0407)  loss_objectness: 0.0008 (0.0025)  loss_rpn_box_reg: 0.0011
(0.0066)  time: 0.3596  data: 0.0656  max mem: 2795
Epoch: [19]  [140/170]  eta: 0:00:10  lr: 0.000000  loss: 0.0500
(0.0729)  loss_classifier: 0.0185 (0.0252)  loss_box_reg: 0.0276
(0.0393)  loss_objectness: 0.0000 (0.0023)  loss_rpn_box_reg: 0.0008
(0.0060)  time: 0.3646  data: 0.0571  max mem: 2795
Epoch: [19]  [160/170]  eta: 0:00:03  lr: 0.000000  loss: 0.0542
(0.0725)  loss_classifier: 0.0211 (0.0252)  loss_box_reg: 0.0287
(0.0392)  loss_objectness: 0.0005 (0.0022)  loss_rpn_box_reg: 0.0015
(0.0059)  time: 0.3575  data: 0.0616  max mem: 2795
Epoch: [19]  [169/170]  eta: 0:00:00  lr: 0.000000  loss: 0.0438
(0.0719)  loss_classifier: 0.0181 (0.0250)  loss_box_reg: 0.0253
(0.0388)  loss_objectness: 0.0002 (0.0022)  loss_rpn_box_reg: 0.0013
(0.0059)  time: 0.3611  data: 0.0597  max mem: 2795
Epoch: [19] Total time: 0:01:01 (0.3631 s / it)
mAP:0.468
     AP at IoU level [0.50]: 0.715
     AP at IoU level [0.55]: 0.698
     AP at IoU level [0.60]: 0.677
     AP at IoU level [0.65]: 0.669
     AP at IoU level [0.70]: 0.632
     AP at IoU level [0.75]: 0.575
     AP at IoU level [0.80]: 0.420
     AP at IoU level [0.85]: 0.245
     AP at IoU level [0.90]: 0.046
     AP at IoU level [0.95]: 0.000
```

Saving the Model

```
from datetime import datetime
# Save model with current date
now = datetime.now()
d = now.strftime("%Y_%b_%d_%Hh_%mm")
PATH = 'model_'+d+'.pt'

torch.save(model.state_dict(), PATH)
```

Evaluate and Predict on Test Set

```
# Get saved model
model_eval = model.load_state_dict(torch.load(PATH))
```

Evaluation:

```
# put the model in evaluation mode
model.eval()

# Evaluate the model
evaluate(model, loader_test, device=device)
```

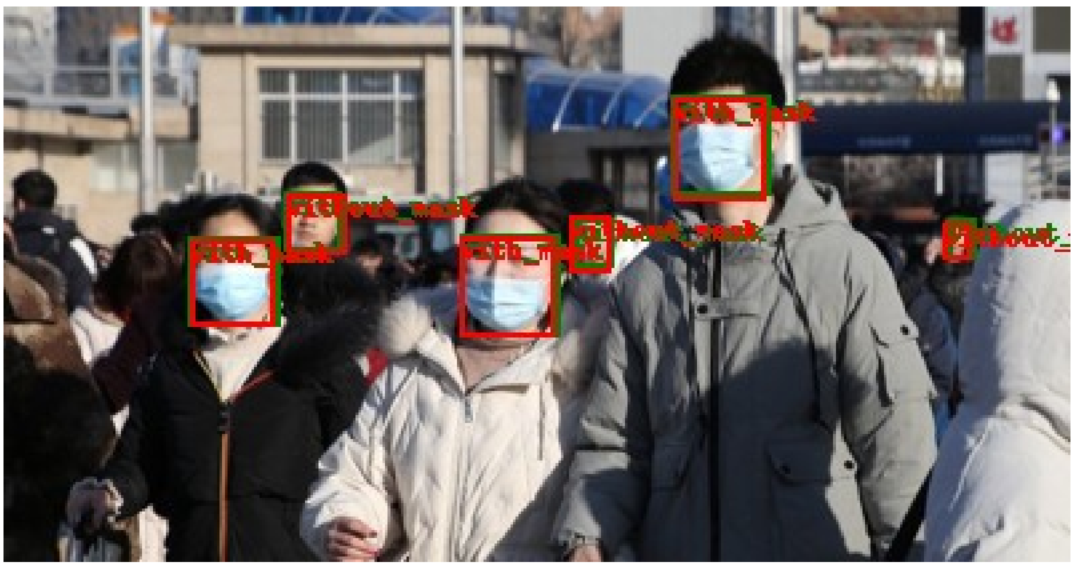
```
mAP:0.419
     AP at IoU level [0.50]: 0.645
     AP at IoU level [0.55]: 0.639
     AP at IoU level [0.60]: 0.629
     AP at IoU level [0.65]: 0.611
     AP at IoU level [0.70]: 0.566
     AP at IoU level [0.75]: 0.485
     AP at IoU level [0.80]: 0.370
     AP at IoU level [0.85]: 0.196
     AP at IoU level [0.90]: 0.044
     AP at IoU level [0.95]: 0.002

(0.41878714099522474,
 [0.6450177478928426,
  0.6389704014059845,
  0.629120241522798,
  0.6111861989017132,
  0.5657634863451461,
  0.48528152182043605,
  0.37007749023925945,
  0.19647219081539297,
  0.044251811806310576,
  0.0017303192023640987])
```

Test prediction on random image.

```
# Make prediction on random image
n = randint(0, dataset_test.len)
img, target = dataset_test[n]
with torch.no_grad():
```

```
    prediction = model([img.to(device)])[0]

# Non max suppression to reduce the number of bounding boxes
nms_prediction = apply_nms(prediction, iou_thresh=0.5)
# Remove low score boxes below score_thresh
filtered_prediction = remove_low_score_bb(nms_prediction, 
score_thresh=0.3)

# Draw bounding boxes
draw_bounding_boxes(img.detach().cpu(), target=target, 
prediction=filtered_prediction)
```

400x208



Evaluation from coco tools

```
from engine import evaluate as eval
eval(model, loader_test, device=device)
```

```
creating index...
index created!
Test:  [ 0/38]  eta: 0:00:07  model_time: 0.1497 (0.1497)  
evaluator_time: 0.0055 (0.0055)  time: 0.2033  data: 0.0461  max mem: 
2795
Test:  [37/38]  eta: 0:00:00  model_time: 0.1278 (0.1290)  
evaluator_time: 0.0088 (0.0208)  time: 0.2180  data: 0.0592  max mem: 
2795
Test: Total time: 0:00:08 (0.2127 s / it)
Averaged stats: model_time: 0.1278 (0.1290)  evaluator_time: 0.0088 
(0.0208)
Accumulating evaluation results...
DONE (t=0.07s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
```

```
maxDets=100 ] = 0.470
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.757
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.519
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.347
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.694
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.799
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.277
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.518
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.540
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.422
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.760
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.823

<coco_eval.CocoEvaluator at 0x7fc2084b6e90>
```

VGG19 Approach

How to create a Deep Learning face mask classifier for COVID-19 in public spaces

Introduction

The CDC continues to monitor the spread of COVID-19 and advises people who are completely vaccinated as well as those who are not fully vaccinated to wear face masks. When visiting the doctor's office, hospitals, or long-term care institutions, the CDC recommends wearing masks and keeping a safe distance.

Manually monitoring people entering such institutions is tedious and requires workforce. In this tutorial, we will learn how we can automate this process through deep learning techniques which will automatically detect people not wearing masks to prevent their entry.

Creating the mask detection deep learning model

We will now look into building a Deep Learning model to predict (detect) if a person is violating the rules by not wearing a mask in public spaces.

Step 1: Importing the necessary Python libraries

```
import numpy as np  # linear algebra
import cv2 # opencv
import matplotlib.pyplot as plt # image plotting
# keras
from keras import Sequential
from keras.layers import Flatten, Dense
from keras.applications.vgg19 import VGG19
from keras.applications.vgg19 import preprocess_input
from keras.preprocessing.image import ImageDataGenerator
```

Step 2: Getting the data

For the training data, we are using the face mask detection data from here. The dataset contains 12 thousand images divided into Test, Train, and Validation sets which were scraped from Google and the CelebFace dataset created by Jessica Li.

```
# Load train and test set
train_dir = "/content/FaceMaskDetection12k/Train"
test_dir = "/content/FaceMaskDetection12k/Test"
val_dir = "/content/FaceMaskDetection12k/Validation"
```

Step 3: Reading a sample image and performing face detection

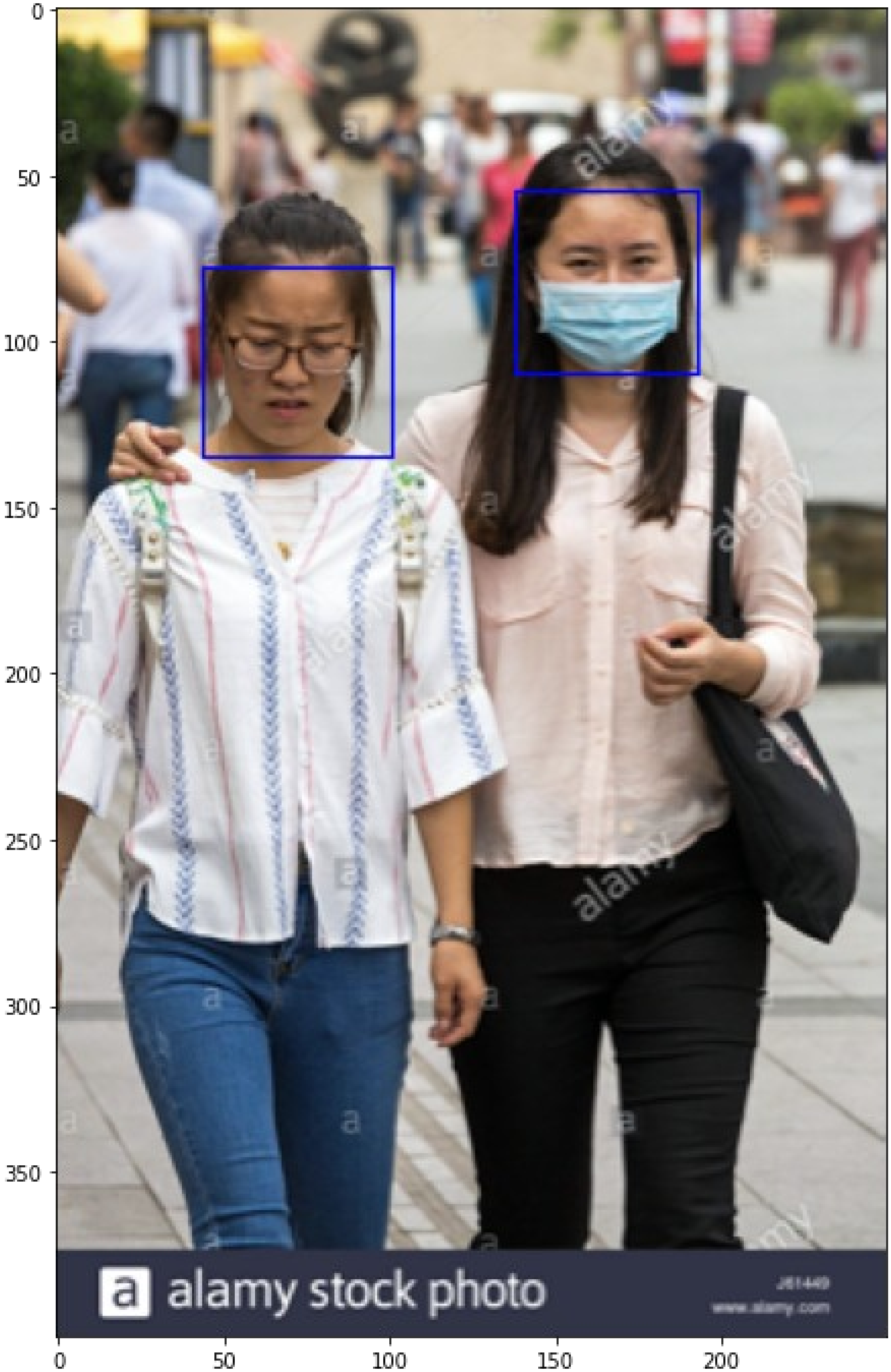
We will now read in a sample image from a busy airport and perform face detection using haar cascade classifier. The Haar cascade classifier, originally known as the Viola-Jones Face Detection Technique is a object detection algorithm for detecting faces in images or real-time video.

Viola and Jones proposed edge or line detection features in their research paper "Rapid Object Detection using a Boosted Cascade of Simple Features," published in 2001. The algorithm is given a large number of positive photos with faces and a large number of negative images with no faces. The model developed as a result of this training can be found in the OpenCV GitHub repository.

```
# Read a sample image
img =
cv2.imread("../input/face-mask-detection/images/maksssksksss352.png")

# Keep a copy of coloured image
orig_img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)  # colored output
image

# Convert image to grayscale
img = cv2.cvtColor(img, cv2.IMREAD_GRAYSCALE)

# loading haarcascade_frontalface_default.xml
face_detection_model = cv2.CascadeClassifier("../input/haar-cascades-
for-face-detection/haarcascade_frontalface_default.xml")
```

```python
# detect faces in the given image
return_faces = face_detection_model.detectMultiScale(
    img, scaleFactor=1.08, minNeighbors=4
)  # returns a list of (x,y,w,h) tuples

# plotting the returned values
for (x, y, w, h) in return_faces:
    cv2.rectangle(orig_img, (x, y), (x + w, y + h), (0, 0, 255), 1)

plt.figure(figsize=(12, 12))
plt.imshow(orig_img)  # display the image
```

```
<matplotlib.image.AxesImage at 0x7fb4fa097450>
```


0
50
100
150
200
250
300
350
0
50
100
150
200
alamy stock photo
www.alamy.com

Step 4: Data preprocessing for building the mask detection Keras model

We will now pass our datasets into Keras ImageDataGenerator() to perform some preliminary data augmentation steps such as rescaling.

```
# Data preprocessing
# Train data
datagenerator = ImageDataGenerator(
    rescale=1.0 / 255, horizontal_flip=True, zoom_range=0.2,
shear_range=0.2
)
train_generator = datagenerator.flow_from_directory(
    directory=train_dir, target_size=(128, 128),
class_mode="categorical", batch_size=32
)

# Validation data
val_generator = datagenerator.flow_from_directory(
    directory=val_dir, target_size=(128, 128),
class_mode="categorical", batch_size=32
)

# Test data
test_generator = datagenerator.flow_from_directory(
    directory=val_dir, target_size=(128, 128),
class_mode="categorical", batch_size=32
)
```

```
Found 10000 images belonging to 2 classes.
Found 800 images belonging to 2 classes.
Found 800 images belonging to 2 classes.
```

Step 5: Create the mask detection transfer learning model using Keras

We are building the deep learning classifer using the VGG19 transfer learning model. The VGG19 model is the successor of AlexNet, a variation of the VGG model named after the group named as Visual Geometry Group at Oxford which created it. It is a deep CNN consisting of 19 layers (16 convolution layers, 3 Fully connected layer, 5 MaxPool layers and 1 SoftMax layer) used to classify images.

It has been trained on ImageNet, a picture database with 14,197,122 images structured according to the WordNet hierarchy.

VGG19 Architecture

```
# Initializing the VGG19 model
vgg19_model = VGG19(weights="imagenet", include_top=False,
input_shape=(128, 128, 3))

for layer in vgg19_model.layers:
    layer.trainable = False
```

```python
# Initialize a sequential model
model = Sequential()
model.add(vgg19_model)
model.add(Flatten())
model.add(Dense(2, activation="sigmoid"))
model.summary()

# Compiling the model
model.compile(optimizer="adam", loss="categorical_crossentropy",
metrics="accuracy")
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80142336/80134624 [==============================] - 1s 0us/step
80150528/80134624 [==============================] - 1s 0us/step
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg19 (Functional)           (None, 4, 4, 512)         20024384
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
_________________________________________________________________
dense (Dense)                (None, 2)                 16386
=================================================================
Total params: 20,040,770
Trainable params: 16,386
Non-trainable params: 20,024,384
_________________________________________________________________
```

Step 6: Train the model

We will now train our neural network model for 20 epochs.

```python
# Fit the model on train data along with validation data
model_history = model.fit_generator(
    generator=train_generator,
    steps_per_epoch=len(train_generator) // 32,
    epochs=20,
    validation_data=val_generator,
    validation_steps=len(val_generator) // 32,
)
```

```
/opt/conda/lib/python3.7/site-packages/keras/engine/training.py:1972: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
```

```
Epoch 1/20

14:36:57.564987: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369]
Loaded cuDNN version 8005

9/9 [==============================] - 11s 370ms/step - loss: 0.5636 -
accuracy: 0.7465
Epoch 2/20
9/9 [==============================] - 3s 348ms/step - loss: 0.2521 -
accuracy: 0.9167
Epoch 3/20
9/9 [==============================] - 3s 389ms/step - loss: 0.1610 -
accuracy: 0.9444
Epoch 4/20
9/9 [==============================] - 3s 344ms/step - loss: 0.1345 -
accuracy: 0.9514
Epoch 5/20
9/9 [==============================] - 3s 324ms/step - loss: 0.1158 -
accuracy: 0.9688
Epoch 6/20
9/9 [==============================] - 3s 348ms/step - loss: 0.1159 -
accuracy: 0.9618
Epoch 7/20
9/9 [==============================] - 3s 326ms/step - loss: 0.1098 -
accuracy: 0.9722
Epoch 8/20
9/9 [==============================] - 3s 353ms/step - loss: 0.1017 -
accuracy: 0.9688
Epoch 9/20
9/9 [==============================] - 3s 311ms/step - loss: 0.0774 -
accuracy: 0.9757
Epoch 10/20
9/9 [==============================] - 3s 314ms/step - loss: 0.0698 -
accuracy: 0.9826
Epoch 11/20
9/9 [==============================] - 3s 370ms/step - loss: 0.0733 -
accuracy: 0.9792
Epoch 12/20
9/9 [==============================] - 3s 285ms/step - loss: 0.0676 -
accuracy: 0.9757
Epoch 13/20
9/9 [==============================] - 3s 293ms/step - loss: 0.0744 -
accuracy: 0.9792
Epoch 14/20
9/9 [==============================] - 3s 313ms/step - loss: 0.0690 -
accuracy: 0.9826
Epoch 15/20
9/9 [==============================] - 3s 280ms/step - loss: 0.0955 -
accuracy: 0.9618
Epoch 16/20
9/9 [==============================] - 3s 295ms/step - loss: 0.0488 -
```

```
accuracy: 0.9861
Epoch 17/20
9/9 [==============================] - 3s 280ms/step - loss: 0.0519 -
accuracy: 0.9826
Epoch 18/20
9/9 [==============================] - 3s 284ms/step - loss: 0.0456 -
accuracy: 0.9931
Epoch 19/20
9/9 [==============================] - 2s 268ms/step - loss: 0.0581 -
accuracy: 0.9792
Epoch 20/20
9/9 [==============================] - 3s 284ms/step - loss: 0.0607 -
accuracy: 0.9722
```

Step 7: Evaluate the model performance on test set

```
# Evaluate model performance on test data
model_loss, model_acc = model.evaluate(test_generator)
print("Model has a loss of %.2f and accuracy %.2f%%" % (model_loss,
model_acc*100))
```

```
25/25 [==============================] - 9s 350ms/step - loss: 0.0678
- accuracy: 0.9775
Model has a loss of 0.07 and accuracy 97.75%
```

Step 8: Save the model

We can also choose to save the trained model as a h5 file for future use.

```
model.save('data/saved_model.h5')
```

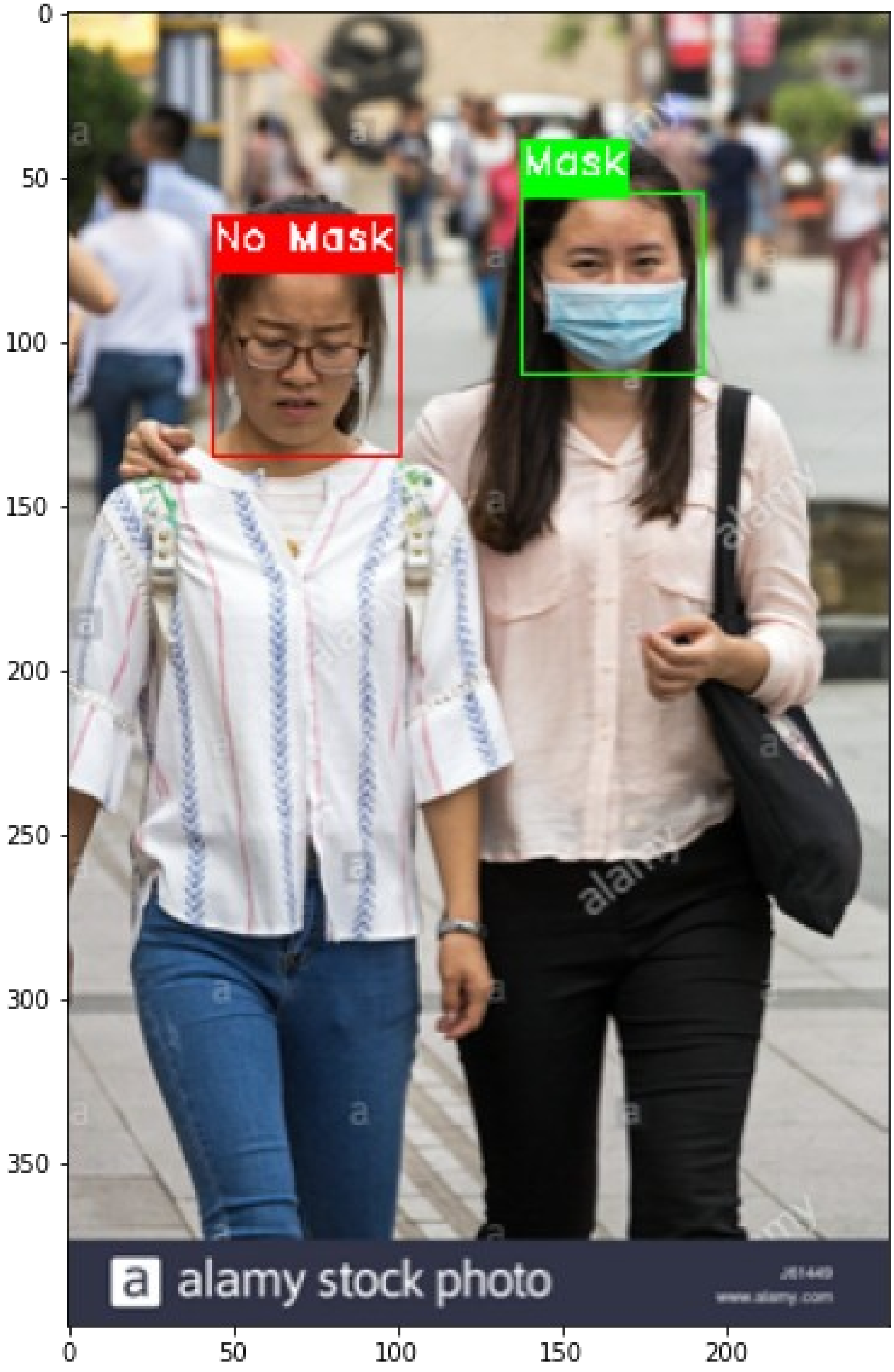
Step 9: Test the model on the sample image

We will now test the trained model on our use case for detecting faces and masks for a group of people. We take the detected face crops of the faces detected in the image and then predict the mask or no mask using the model trained.

```
# label for mask detection
mask_det_label = {0: "Mask", 1: "No Mask"}
mask_det_label_colour = {0: (0, 255, 0), 1: (255, 0, 0)}
pad_y = 1  # padding for result text

main_img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)  # colored output
image

# For detected faces in the image
for i in range(len(return_faces)):
    (x, y, w, h) = return_faces[i]
    cropped_face = main_img[y : y + h, x : x + w]
    cropped_face = cv2.resize(cropped_face, (128, 128))
    cropped_face = np.reshape(cropped_face, [1, 128, 128, 3]) / 255.0
    mask_result = model.predict(cropped_face)  # make model prediction
```

```python
    print_label = mask_det_label[mask_result.argmax()] # get mask/no
mask based on prediction
    label_colour = mask_det_label_colour[mask_result.argmax()] # green
for mask, red for no mask

    # Print result
    (t_w, t_h), _ = cv2.getTextSize(
        print_label, cv2.FONT_HERSHEY_SIMPLEX, 0.4, 1
    )  # getting the text size

    cv2.rectangle(
        main_img,
        (x, y + pad_y),
        (x + t_w, y - t_h - pad_y - 6),
        label_colour,
        -1,
    )  # draw rectangle

    cv2.putText(
        main_img,
        print_label,
        (x, y - 6),
        cv2.FONT_HERSHEY_DUPLEX,
        0.4,
        (255, 255, 255), # white
        1,
    )  # print text

    cv2.rectangle(
        main_img,
        (x, y),
        (x + w, y + h),
        label_colour,
        1,
    )  # draw bounding box on face

plt.figure(figsize=(10, 10))
plt.imshow(main_img)  # display image
```

```
<matplotlib.image.AxesImage at 0x7fb4b8b52910>
```



We can see that the model is correctly detecting faces and classifying them as mask and no mask.