به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین سوم

نام و نام خانوادگی	**محمد پویا افشاری – علیرضا اسمعیل زاده**
شماره دانشجویی	**810198351-810198577**
تاریخ ارسال گزارش	**1402.09.17**

فهرست

شکل‌ها

پاسخ ۱ – SAM

۱-۱. آماده سازی مجموعه داده

۱-۱-۱ . افزودن کتاب خانه ها

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from tensorflow import keras
import tensorflow as tf

from keras.preprocessing.image import ImageDataGenerator
import keras.utils as image
import os
import seaborn as sns

from keras.callbacks import ModelCheckpoint, EarlyStopping
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.utils import compute_class_weight
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc
import shutil
from google.colab import drive
from google.colab import files
import random
```

۱-۱-۲ . جمع آوری داده

```
uploaded = files.upload()

!mkdir -p ~/.kaggle
!mv kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
!kaggle datasets download -d franciscoescobar/satellite-images-of-water-bodies
!unzip -q satellite-images-of-water-bodies.zip
```

```
Saving kaggle.json to kaggle.json
Downloading satellite-images-of-water-bodies.zip to /content
 98% 243M/247M [00:02<00:00, 97.0MB/s]
100% 247M/247M [00:02<00:00, 101MB/s]
```

داده مورد نیاز را از Kaggle دانلود می‌کنیم و داده zip شده ی آن را unzip می‌کنیم.

۳-۱-۱ . resize تصاویر

```
def img_resize(image, y_dim, x_dim):
    resized_img = cv2.resize(image, (y_dim,x_dim))
    return resized_img
```

تعریف تابع برای resize

```
from google.colab import drive
drive.mount('/content/drive')
```

کد برای mount کردن گوگل درایو برای ذخیره دیتا ها

```
image_path = "/content/Water Bodies Dataset/Images/*.jpg"
mask_path = "/content/Water Bodies Dataset/Masks/*.jpg"
```

تعریف مسیر فایل های تصاویر به همراه ماسک ها

```
import glob
image_names = sorted(glob.glob(image_path), key=lambda x: x.split('.')[0])
mask_names = sorted(glob.glob(mask_path), key=lambda x: x.split('.')[0])
```

خواندن فایل ها از تعریف شده با استفاده از کتاب خانه glob

```
train_images_array = []

for image in image_names:
    img = cv2.imread(image, 0)
    img = img_resize(img, 256, 256)
    train_images_array.append(img)

images = np.array(train_images_array)
images.shape
```

خواندن محتویات هر عکس به صورت gray با استفاده از کتاب خانه cv سپس resize کردن به ابعاد ۲۵۶ در ۲۵۶ و تبدیل آن به آرایه numpy

```
print(images.shape)
```

```
(2841, 256, 256)
```

گرفتن image shape ها برای اطمینان از درست بودن دیتا ها

```
mask_images_array = []

for mask in mask_names:
    msk = cv2.imread(mask, 0)
    msk = img_resize(msk, 256, 256)
    mask_images_array.append(msk)

masks = np.array(mask_images_array)
```

خواندن محتویات هر ماسک به صورت gray با استفاده از کتاب خانه cv سپس resize کردن به ابعاد ۲۵۶ در ۲۵۶ و تبدیل آن به آرایه numpy

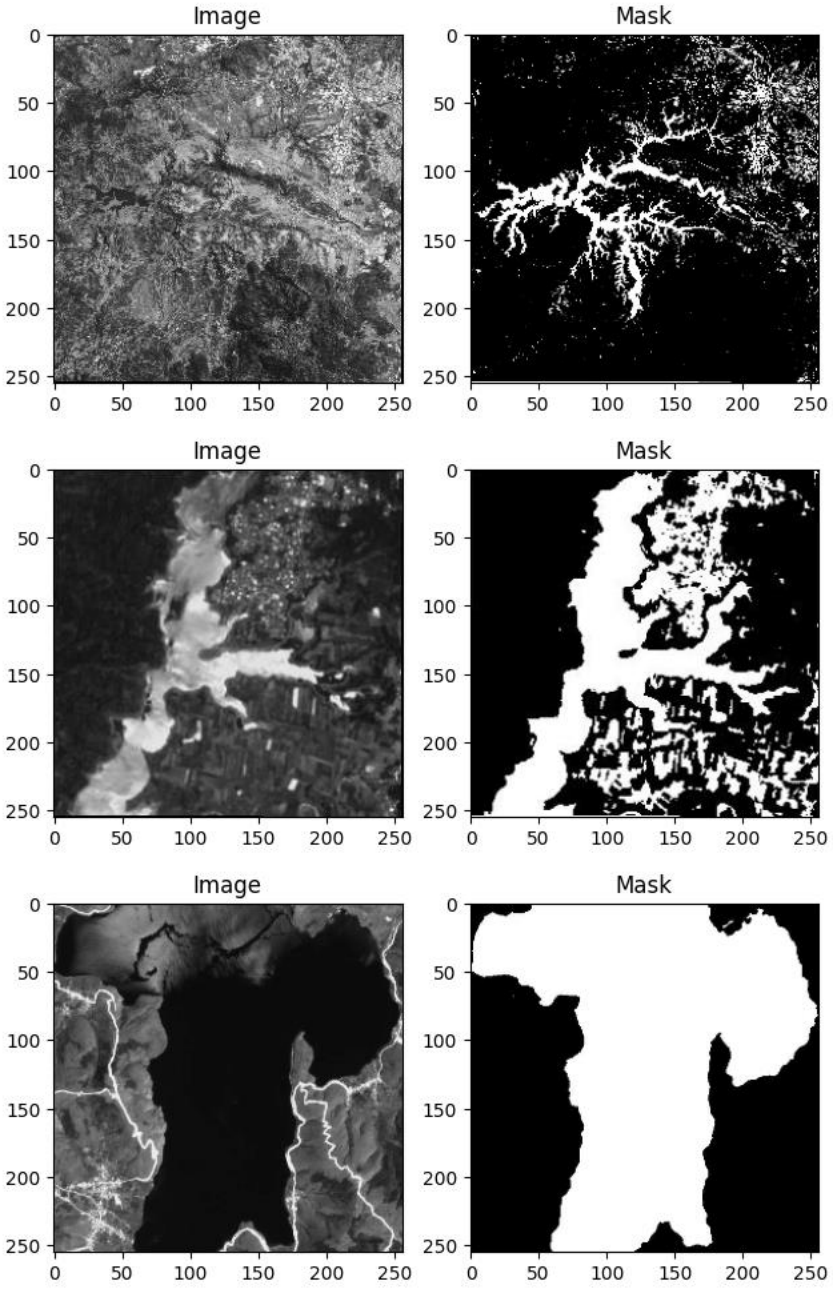
```
print(images.shape, masks.shape)
```

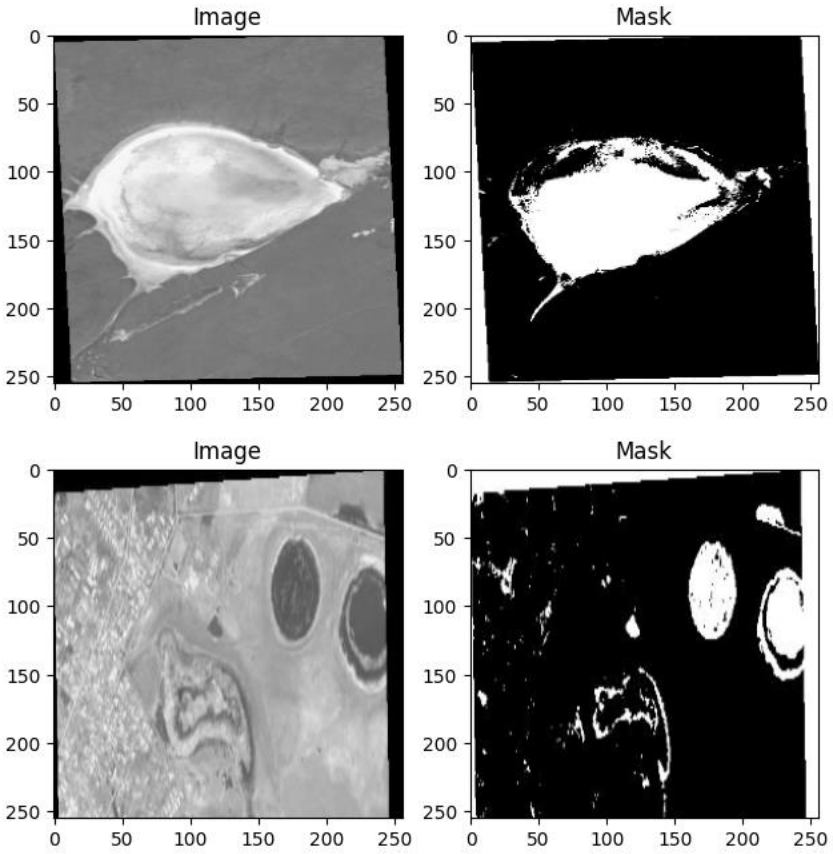
```
(2841, 256, 256) (2841, 256, 256)
```

گرفتن shape ها برای اطمینان از درست بودن دیتا ها

۴-۱-۱. نمونه تصاویر

```
for i in range(5):
    plt.subplot(1, 2, 1)
    plt.imshow(images[i], cmap='gray')
    plt.title('Image')

    plt.subplot(1, 2, 2)
    plt.imshow(masks[i], cmap='gray')
    plt.title('Mask')

    plt.tight_layout()
    plt.show()
```





شکل(1-1): نمونه‌ای تصاویر در کنار ماسک های متناظر

نمایش ۵ تصویر نمونه به همراه ماسک به صورت gray با استفاده از کتاب خانه matplotlib

۵-۱-۱. تقسیم داده به دو بخش آموزش و ارزیابی

```
X_train, X_test, y_train, y_test = train_test_split(images, masks, test_size = 0.1, random_state = 3)

print(X_train.shape, X_test.shape)
```

```
(2556, 256, 256) (285, 256, 256)
```

تقسیم داده به دو بخش آموزش و ارزیابی با نسبت ۹ به ۱

۱-۲. بارگذاری مدل

۱-۲-۱. آماده سازی دیتاست

```
!pip install datasets
```

نصب کتاب خانه دیتاست

```
from datasets import Dataset
from PIL import Image

# Convert the NumPy arrays to Pillow images and store them in a dictionary
dataset_dict = {
    "image": [Image.fromarray(img) for img in images],
    "label": [Image.fromarray(mask) for mask in masks],
```

```
}

# Create the dataset using the datasets.Dataset class
dataset = Dataset.from_dict(dataset_dict)
```

برای راحتی کار با داده تصویری ابتدا آرایه numpy را به Image کتاب خانه Pillow تبدیل می‌کنیم سپس آن را به صورت دیکشنری در آبجکت dataset ذخیره می‌کنیم.

```
def get_bounding_box(ground_truth_map):
  if len(ground_truth_map.shape) == 3:
    ground_truth_map = ground_truth_map[:, :, 0]
  # get bounding box from mask
  y_indices, x_indices = np.where(ground_truth_map > 0)
  x_min, x_max = np.min(x_indices), np.max(x_indices)
  y_min, y_max = np.min(y_indices), np.max(y_indices)
  # add perturbation to bounding box coordinates
  H, W = ground_truth_map.shape
  x_min = max(0, x_min - np.random.randint(0, 20))
  x_max = min(W, x_max + np.random.randint(0, 20))
  y_min = max(0, y_min - np.random.randint(0, 20))
  y_max = min(H, y_max + np.random.randint(0, 20))
  bbox = [x_min, y_min, x_max, y_max]

  return bbox
```

تابع get_bounding_box را برای ایجاد bounding box از روی ماسک ها تعریف می‌کنیم. تا به عنوان prompt استفاده شود.

```
from torch.utils.data import Dataset

class SAMDataset(Dataset):
  """
  This class is used to create a dataset that serves input images and masks.
  It takes a dataset and a processor as input and overrides the __len__ and __getitem__ methods of the
Dataset class.
  """
  def __init__(self, dataset, processor):
    self.dataset = dataset
    self.processor = processor

  def __len__(self):
    return len(self.dataset)

  def __getitem__(self, idx):
    item = self.dataset[idx]
    image = item["image"]
    ground_truth_mask = np.array(item["label"])
    # get bounding box prompt
    prompt = get_bounding_box(ground_truth_mask)
```

```
    # prepare image and prompt for the model
    inputs = self.processor(image, input_boxes=[[prompt]], return_tensors="pt")

    # remove batch dimension which the processor adds by default
    inputs = {k:v.squeeze(0) for k,v in inputs.items()}

    # add ground truth segmentation
    inputs["ground_truth_mask"] = ground_truth_mask

    return inputs
```

تابع get_bounding_box را برای ایجاد bounding box از روی ماسک ها تعریف می‌کنیم. کلاس SAMDataset را برای استفاده در مدل SAM با فیلد های دیتاست که در بخش قبل ایجاد کردیم و processor که SamProcessor به عنوان ورودی تعریف می‌کنیم همچنین توابع __len__ برای گرفتن اندازه دیتاست با استفاده از تابع len و همچنین تابع __getitem__ برای برگرداندن هر فیلد هنگام فراخوانی ایندکس آبجکت را پیاده سازی می‌کنیم.

```
# Initialize the processor
from transformers import SamProcessor
processor = SamProcessor.from_pretrained("facebook/sam-vit-base")
```

ساخت آبجکت SamProcessor با استفاده از کتاب خانه transformers

```
train_dataset = SAMDataset(dataset=dataset, processor=processor)
```

ساخت آبجکت SAMDataset با استفاده از دیتاست و processor تعریف شده در بخش های قبلی

```
example = train_dataset[0]
for k,v in example.items():
  print(k,v.shape)
```

```
pixel_values torch.Size([3, 1024, 1024])
original_sizes torch.Size([2])
reshaped_input_sizes torch.Size([2])
input_boxes torch.Size([1, 4])
ground_truth_mask (256, 256)
```

تست تابع __getitem__ و بررسی shape خروجی آن

```
from torch.utils.data import DataLoader
train_dataloader = DataLoader(train_dataset, batch_size=2, shuffle=True, drop_last=False)
```

ساخت آبجکت با مقادیر دیتاست train_dataloader با استفاده از کتاب خانه torch.utils.data برای استفاده در زمان بارگذاری در مدل

```
batch = next(iter(train_dataloader))
for k,v in batch.items():
  print(k,v.shape)
```

```
ixel_values torch.Size([2, 3, 1024, 1024])
original_sizes torch.Size([2, 2])
reshaped_input_sizes torch.Size([2, 2])
input_boxes torch.Size([2, 1, 4])
ground_truth_mask torch.Size([2, 256, 256])
```

بررسی داده های dataloader و shape آن برای اطمینان از مقداردهی درست قبل از بارگذاری در مدل

```
from transformers import SamModel
```

```
model = SamModel.from_pretrained("facebook/sam-vit-base")

# make sure we only compute gradients for mask decoder
for name, param in model.named_parameters():
  if name.startswith("vision_encoder") or name.startswith("prompt_encoder"):
    param.requires_grad_(False)
```

ساخت مدل SAM و فریز کردن دو پارمتر vision_encoder و prompt_encoder برای حفظ ویژگی های غنی

۱-۳. تقویت داده

در این بخش به کمک albumentations اقدام به Augment کردن دیتا به صورت Flip و ... میکنیم.

```
# Train set transformations
import albumentations as A
from albumentations.pytorch.transforms import ToTensorV2
transform_train = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.2),
    ToTensorV2(p=, label_fields=['labels']))

# Validation set transformations
transform_valid = A.Compose([
    ToTensorV2(p=1)], bbox_params=A.BboxParams(label_fields=['labels']))
```

۱-۴. بهینه‌ساز، متریک و تابع هزینه

```
!pip install -qU monai
```

نصب کتابخانه monai برای استفاده از متریک و تابع هزینه

```
import monai
from monai.losses import DiceCELoss
from monai.metrics import DiceMetric, MeanIoU


from tqdm import tqdm
from statistics import mean
import torch
from torch.nn.functional import threshold, normalize
```

نصب کتابخانه های مورد نیاز

```
from torch.optim import Adam

optimizer = Adam(model.mask_decoder.parameters(), lr=1e-5, weight_decay=0)
seg_loss = monai.losses.DiceCELoss(sigmoid=True, squared_pred=True, reduction='mean')
```

برای تابع هزینه و بهینه ساز از Adam و DiceCELoss استفاده می‌کنیم.

۱-۵. Fine-Tune کردن مدل

۱-۵-۱ . آموزش مدل

```
from tqdm import tqdm
from statistics import mean
import torch
from torch.nn.functional import threshold, normalize

#Training loop
num_epochs = 1

device = "cuda" if torch.cuda.is_available() else "cpu"
model.to(device)

model.train()
for epoch in range(num_epochs):
    epoch_losses = []
    for batch in tqdm(train_dataloader):
      # forward pass
      outputs = model(pixel_values=batch["pixel_values"].to(device),
                      input_boxes=batch["input_boxes"].to(device),
                      multimask_output=False)

      # compute loss
      predicted_masks = outputs.pred_masks.squeeze(1)
      ground_truth_masks = batch["ground_truth_mask"].float().to(device)
      loss = seg_loss(predicted_masks, ground_truth_masks.unsqueeze(1))

      # backward pass (compute gradients of parameters w.r.t. loss)
      optimizer.zero_grad()
      loss.backward()

      # optimize
      optimizer.step()
      epoch_losses.append(loss.item())

    print(f'EPOCH: {epoch}')
    print(f'Mean loss: {mean(epoch_losses)}')
```

```
100%|| 1421/1421 [25:43<00:00,  1.09s/it]
EPOCH: 0
Mean loss: -1798159.708056118
```

داده را در یک ایپاک آموزش می دهیم. اما تابع loss مقدار 1798159.708056118- را نشان می‌دهد.

۱-۵-۲ . ذخیره مدل

```
torch.save(model.state_dict(), "/content/drive/MyDrive/Colab Notebooks/models/mito_model_checkpoint.pth")
```

ذخیره مدل آموزش دیده بر روی گوگل درایو

۱-۶. ازیابی نتایج

۱-۶-۱ . افزودن کتاب‌خانه ها

```
from transformers import SamModel, SamConfig, SamProcessor
import torch
```

۱-۶-۲ . لود کردن مدل ذخیره شده

```
# Load the model configuration
model_config = SamConfig.from_pretrained("facebook/sam-vit-base")
processor = SamProcessor.from_pretrained("facebook/sam-vit-base")

# Create an instance of the model architecture with the loaded configuration
my_mito_model = SamModel(config=model_config)
#Update the model by loading the weights from saved file.
my_mito_model.load_state_dict(torch.load("/content/drive/MyDrive/Colab
Notebooks/models/mito_model_checkpoint.pth"))
```

ذخیره مدل ذخیره شده را از گوگل کلود لود می‌کنیم و کانفیگ های مربوط به مدل SAM را نیز بر هنگام لود کردن مدل به عنوان کانفیگ می‌دهیم.

```
# set the device to cuda if available, otherwise use cpu
device = "cuda" if torch.cuda.is_available() else "cpu"
my_mito_model.to(device)
```

```
SamModel(
  (shared_image_embedding): SamPositionalEmbedding()
  (vision_encoder): SamVisionEncoder(
    (patch_embed): SamPatchEmbeddings(
      (projection): Conv2d(3, 768, kernel_size=(16, 16), stride=(16, 16))
    )
    (layers): ModuleList(
      (0-11): 12 x SamVisionLayer(
        (layer_norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
        (attn): SamVisionAttention(
          (qkv): Linear(in_features=768, out_features=2304, bias=True)
          (proj): Linear(in_features=768, out_features=768, bias=True)
        )
        (layer_norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
        (mlp): SamMLPBlock(
          (lin1): Linear(in_features=768, out_features=3072, bias=True)
          (lin2): Linear(in_features=3072, out_features=768, bias=True)
          (act): GELUActivation()
        )
      )
    )
    (neck): SamVisionNeck(
      (conv1): Conv2d(768, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (layer_norm1): SamLayerNorm()
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
```

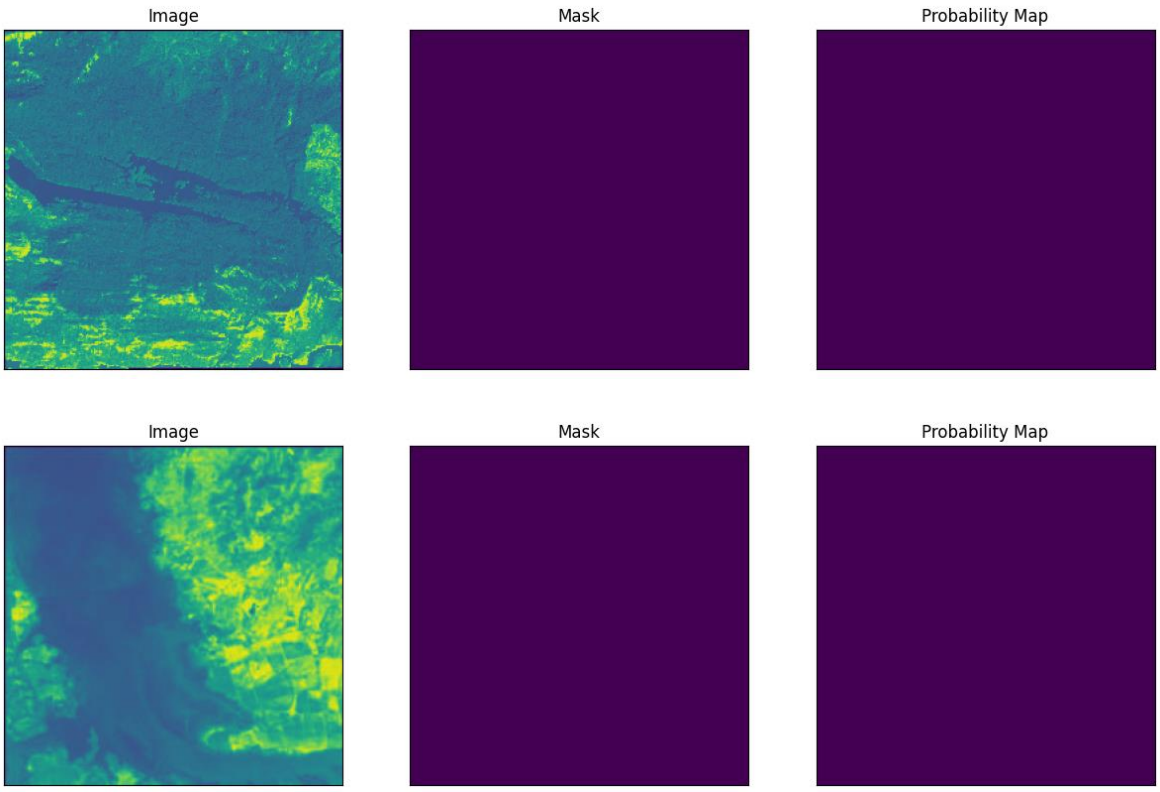
```
...
        (0): Linear(in_features=256, out_features=256, bias=True)
      )
    )
  )
)
```

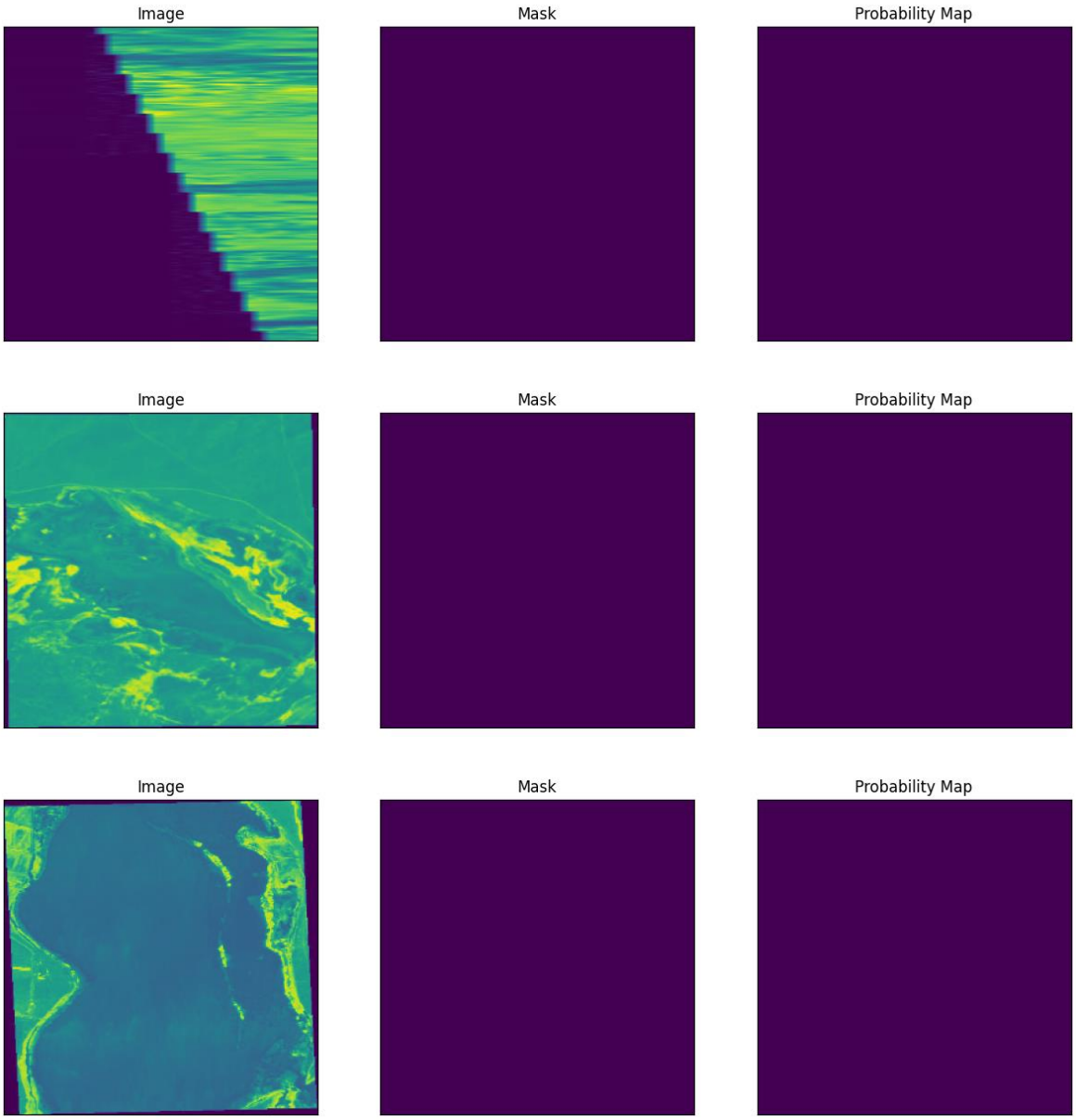
استفاده از GPU و بارگذاری مدل بر روی GPU برای افزایش performance

۳-۶-۱ . اجرای داده ازیابی بر روی مدل

```
import numpy as np
import random
import torch
import matplotlib.pyplot as plt

def show_random_image_with_mask():
  # let's take a random training example
  idx = random.randint(0, images.shape[0]-1)

  # load image
  test_image = dataset[idx]["image"]

  # get box prompt based on ground truth segmentation map
  ground_truth_mask = np.array(dataset[idx]["label"])
  prompt = get_bounding_box(ground_truth_mask)

  # prepare image + box prompt for the model
  inputs = processor(test_image, input_boxes=[[prompt]], return_tensors="pt")

  # Move the input tensor to the GPU if it's not already there
  inputs = {k: v.to(device) for k, v in inputs.items()}

  my_mito_model.eval()

  # forward pass
  with torch.no_grad():
      outputs = my_mito_model(**inputs, multimask_output=False)

  # apply sigmoid
  medsam_seg_prob = torch.sigmoid(outputs.pred_masks.squeeze(1))
  # convert soft mask to hard mask
  medsam_seg_prob = medsam_seg_prob.cpu().numpy().squeeze()
  medsam_seg = (medsam_seg_prob > 0.5).astype(np.uint8)


  fig, axes = plt.subplots(1, 3, figsize=(15, 5))

  # Plot the first image on the left
  axes[0].imshow(np.array(test_image))  # Assuming the first image is grayscale
  axes[0].set_title("Image")
```

```
 # Plot the second image on the right
 axes[1].imshow(medsam_seg)  # Assuming the second image is grayscale
 axes[1].set_title("Mask")

 # Plot the second image on the right
 axes[2].imshow(medsam_seg_prob)  # Assuming the second image is grayscale
 axes[2].set_title("Probability Map")

 # Hide axis ticks and labels
 for ax in axes:
     ax.set_xticks([])
     ax.set_yticks([])
     ax.set_xticklabels([])
     ax.set_yticklabels([])

 # Display the images side by side
 plt.show()

for i in range(5):
 show_random_image_with_mask()
```





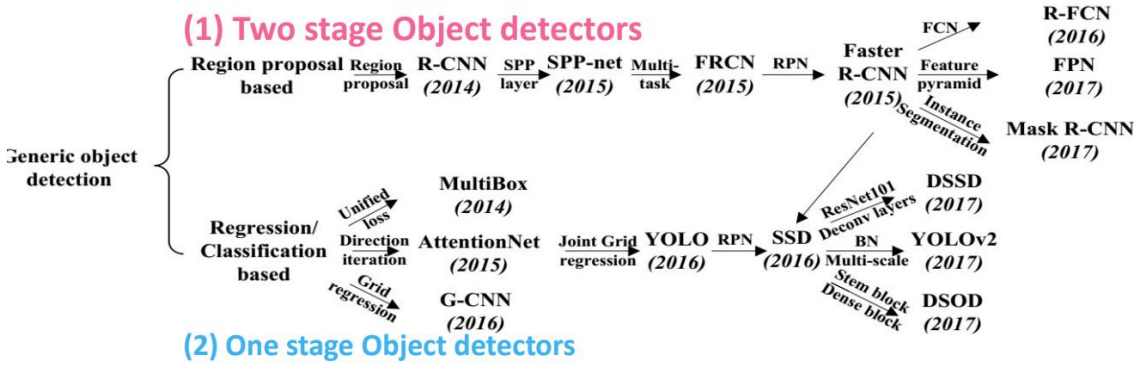
شکل(۲-۱): تصاویر ارزیابی همراه ماسک واقعی، ماسک پیشبینی

۵ نمونه از داده های ارزیابی را به عنوان ورودی به مدل داده شده و خروجی ماسک های آن را نمایش می دهیم به دلیل loss بالا هنگام آموزش مدل تصاویر ماسک های واقعی و پیشبینی شده از عملکرد خوبی برخوردار نیستند.

پاسخ ۲ – آشنایی و پیاده سازی مدل Faster RCNN

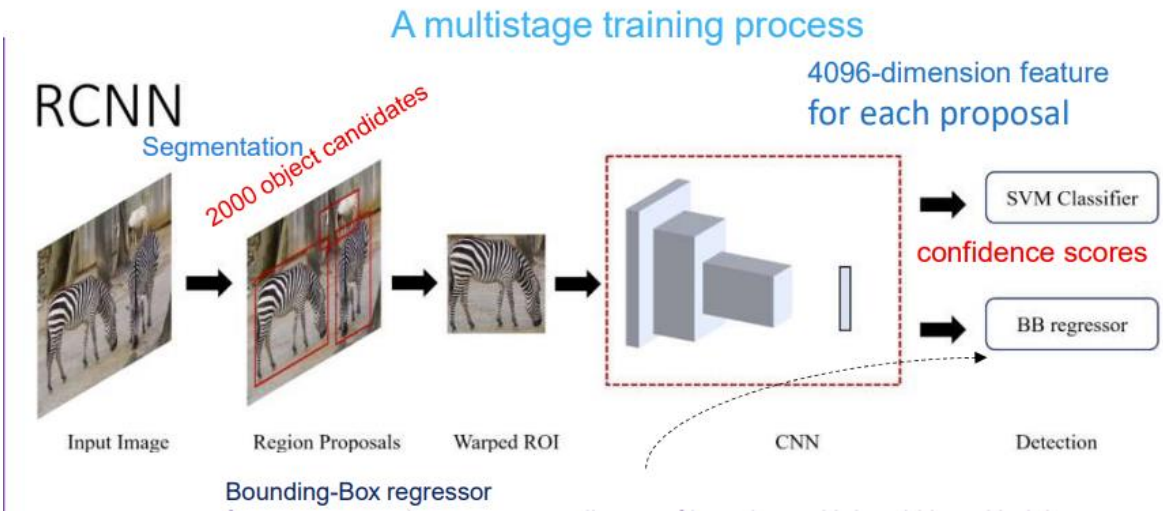
۲-۱. توضیحات مدل ها

به طور کلی از مدل های RCNN برای حوزه‌ی Object detection و مشخص کردن اشیا با استفاده از Bounding box ها قابل تصور است.



شکل (2-1): 0—1تاریخچه مدل های **RCNN**

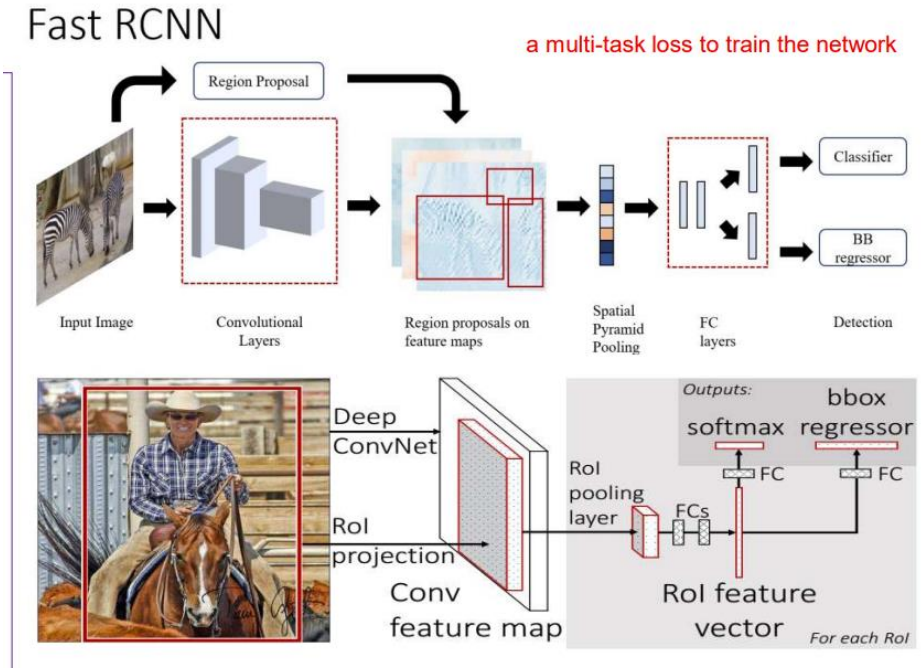
- RCNN:



شکل (2-2): 0—3مدل **RCNN**

به طور کلی معماری این مدل از دو بخش تشکیل می‌شود. بخش اول تولید کننده ویژگی‌هایی خواهد بود که در آن یک Object منحصر وجود دارد، که با Selective search پیدا می‌شود. در مرحله دوم Region مشخص شده به مدل کانولوشن وارد می‌شود تا به هدف Detection. مرحله اول در RCNN با SVM صورت می‌گیرد و در آن از شبکه استفاده نمی‌شود و کاربرد های پردازش تصویری آن بارز تر است.
همانطور که دیده می‌شود این مدل در 2014 به وجود آمده و بر اساس Region proposal ارایه شده کار می‌کند.

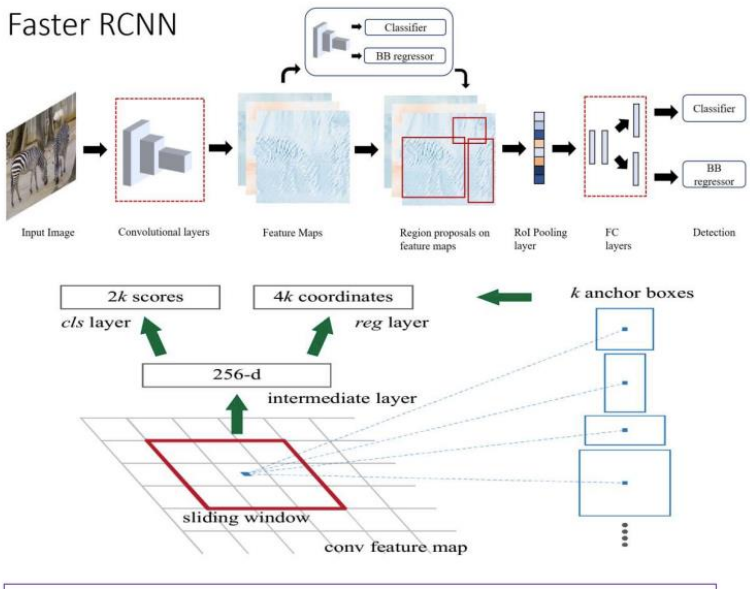
- Fast RCNN



شکل(3-2): مدل **Faster RCNN**

این مدل که در سال 2015 ارائه شد مشکل train شدن جداگانه هر عکس را در RCNN رفع کرد. در این مدل هنوز پروپوزال ایجاد می‌شود ولی در اینجا مدل End to end به همراه CNN و SVM برای پروپوزال ایجاد می‌شود که قسمت پایین عکس بیان همین موضوع است. در اینجا کل عکس به همراه input map ها در شبکه CNN قرار داده می‌شود. در ادامه به کمک Max pooling و Dimentions reduction برای Feature maها سایز کاهش می‌یابد. به کمک دو Fully connected تفکیک پذیری اشیا امکان پذیر می‌شود. در این مدل برخلاف RCNN از L1 به جای L2 برای Loss function of bounding box استفاده شده است.

- Faster RCNN



شکل(4-2): مدل **Faster RCNN**

با وجود افزایش سرعت اما هنوزFast R-CNN هم Cost زیادی به همراه داشت به همین علت مدل Faster R-CNN به ارائه Region Proporsal Network (RPN) این مشکل را بهبود بخشید. به این وسیله در این مدل هر

سایز عکسی به ورودی CNN داده می‌شود و در انتها یک ست از Candidate window تشکیل می‌شود. هر پنجره با Score مشخص شونده‌ی احتمالی از وقوع شی در پنجره خواهد بود. در این مدل بر خلاف مدل های قبل که از Pyramid برای حل مشکل تفاوت اندازه استفاده می‌کردند در اینجا مفهوم Anchor box ایجاد می‌شود. به این وسیله که چندین bounding box با ابعاد مختلف را دریافت می‌کند و در راستای یک شکل آنها را همگام می‌سازد. نتیجه به RPN انتقال می‌یابد و روی ان Classification صورت می‌گیرد. پروپوزال های منتخب به ROI pooling منتقل شده و بعد از آن به وسیله FC تفکیک می‌شود.
در این بحث به کلید واژه های این شبکه پرداخته شد بنابر این نیاز است با جزییات بیشتر توزیح داده شود:

- Convolutional Layer:

از Building block های این مدل است که به این وسیله عکس دریافتی Feature های اشیا در این مشخص میگردد که برای مراحل بعد آماده بشود.

- RPN (Region Proposal Network):

این بخش وظیفه‌ی ارائه پروپوزال در مدل Faster R-CNN را دارد که باید Feature map های تولید شده بخش CNN را کاندید برای Object rejion بکند. این کار به وسیله ی شبکه کوچک شامل (معمولا چندین لایه کانولوشن صورت می‌گیرد.) بر روی Feature map و پیش بینی می‌کند که آیا شی هست داخل بخش یا خیر.

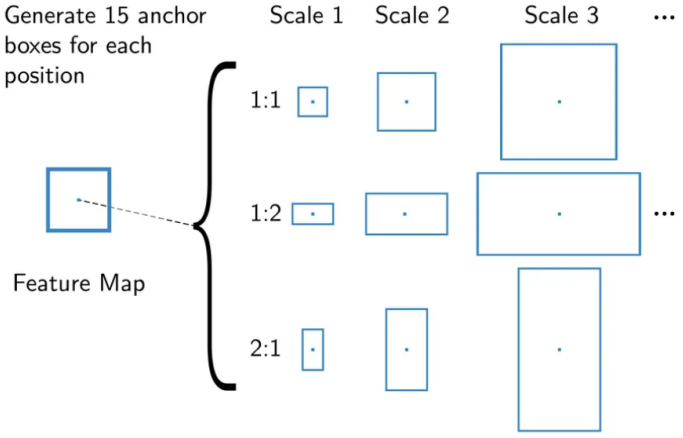
- ROI (Region of Interest) Pooling:

بعد از RPN دراین مرحله Feature map های سایز مشخص Pool می‌شود. به این علت که ممکن است کاندیدا ها در سایز های مختلف باشد که برای ورودی شبکه باید به سایز مناسب Resize بشوند.

- Classification Layer:

این مرحله وضیفه دارد که نواحی داخل هر Proposal را مشخص کند. این بخش که معمولا به ROI وصل می‌شود بر اساس Score های بدست آمده نواحی BB هر شی را تفکیک پذیر و نوع شی یا در صورت نیاز معلوم می‌سازد.

- Anchor Boxes:



شکل(5-2): **Anchor box**.

همان طور که اشاره شد به کمک Anchor box ها جعبه‌ها با اندازه ویژگی‌های مختلف هستند که به جستجوی ویژگی روی تصویر اعمال می‌شوند.

۲-۲ . پیش پردازش

در فاز پیش پردازش نمونه‌های تصاویر را نشان می‌دهیم فرمت خروجی در اندازه 480 در 640 هست. مدل Faster RCNN همان طور که اشاره شد به اندازه ورودی حساس نیست و میتوانیم هر ابعادی بدهیم ولی برای افزایش سرعت یادگیری میتوانیم آن را به اندازه کوچکتر کاهش بدهیم. در یادگیری از هر دو روش استفاده شد و نتایج نسبتا یکسان بدست آمد بنابر این با اندازه اولیه ادامه می‌دهیم ولی هر دو کد را قرار میدهیم.
نکتته قابل توجه آن است که چون فرمت ورودی Pascal VOC XML برای تغییر اندازه باید اندازه در XML فایل هم بروز شود و اندازه Labelهم با نسبت Fraction درستی تغییر یابد.

۲-۲-۱ . کتابخانه

```
from torch.utils.data import DataLoader
from torchvision.models.detection import fasterrcnn_resnet50_fpn
import math
import sys
import time
import torch
import os
from scipy.io import loadmat
from google.colab import drive
import random
import matplotlib.pyplot as plt
from PIL import Image
import xml.etree.ElementTree as ET
import torchvision.models as models
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
import warnings

warnings.filterwarnings('ignore')
drive.mount('/content/drive')
gpu = True

if gpu == True:
    device = torch.device('cuda')
else:
    device = torch.device('cpu')
```

۲-۲-۲ . افزودن دیتاست

```
source_path = '/content/drive/MyDrive/Wildfire Smoke.v1-raw.voc/'
destination_path = '/content/dataset'
!cp -r "$source_path" "$destination_path"
```

۲-۲-۳ . نشان دادن تصاویر نمونه

```


# Paths
xml_folder = '/content/dataset/train'
image_folder = '/content/dataset/train'
```

```
def extract_info(xml_path):
    tree = ET.parse(xml_path)
    root = tree.getroot()

    label = root.find('.//name').text

    size_elem = root.find('.//size')
    width = int(size_elem.find('width').text)
    height = int(size_elem.find('height').text)

    return label, width, height

xml_files = [f for f in os.listdir(xml_folder) if f.endswith('.xml')]

import random
random.seed(42)
selected_files = random.sample(xml_files, 5)

plt.figure(figsize=(20, 3))

for i, xml_file in enumerate(selected_files):
    xml_path = os.path.join(xml_folder, xml_file)

    label, width, height = extract_info(xml_path)

    plt.subplot(1, 5, i + 1)
    plt.imshow(Image.open(os.path.join(image_folder, xml_file.replace('.xml', '.jpg'))))
    plt.title(f"Label: {label}, Size: {width} x {height}")
    plt.axis('off')

plt.show()
```

شکل (6-2): تصاویر نمونه بدست آمده دیتاست

۲-۲-۴ . بدست آوردن تعداد کلاس‌ها و تغییر اندازه

```
def extract_labels(xml_path):
    tree = ET.parse(xml_path)
    root = tree.getroot()
    # Extract label
    label = root.find('.//name').text
    return label
xml_files = [f for f in os.listdir(xml_folder) if f.endswith('.xml')]
unique_labels = set()

for xml_file in xml_files:
    xml_path = os.path.join(xml_folder, xml_file)

    # Extract label
```

```
    label = extract_labels(xml_path)

    # Add label to set
    unique_labels.add(label)

# Number of unique labels
num_unique_labels = len(unique_labels)

print(f"Number of unique labels: {num_unique_labels}")
print("Unique Labels:", unique_labels)
```

```
Mounted at /content/drive
```

```
Number of unique labels: 1
Unique Labels: {'smoke'}
```

```
folders = ['/content/dataset/train', '/content/dataset/valid', '/content/dataset/test']

# def custom_round(number):
#     integer_part = int(number)
#     fraction_part = number - integer_part

#     if fraction_part > 0.5:
#         return integer_part + 1
#     else:
#         return integer_part

# def resize_bounding_box(original_width, original_height, xmin, xmax, ymin, ymax,
target_width, target_height):
#     resize_factor_x = target_width / original_width
#     resize_factor_y = target_height / original_height

#     resized_xmin = custom_round(xmin * resize_factor_x)
#     resized_xmax = custom_round(xmax * resize_factor_x)
#     resized_ymin = custom_round(ymin * resize_factor_y)
#     resized_ymax = custom_round(ymax * resize_factor_y)

#     return resized_xmin, resized_xmax, resized_ymin, resized_ymax

# def resize_image_and_xml(image_path, xml_path):
#     img = Image.open(image_path)
#     img = img.resize(desired_size)

#     tree = ET.parse(xml_path)
#     root = tree.getroot()

#     size_elem = root.find('.//size')

#     xmin = int(root.find('.//xmin').text)
#     xmax = int(root.find('.//xmax').text)
#     ymin = int(root.find('.//ymin').text)
#     ymax = int(root.find('.//ymax').text)
#     width = int(root.find('.//width').text)
#     height = int(root.find('.//height').text)

#     resized_xmin, resized_xmax, resized_ymin, resized_ymax = resize_bounding_box(
#       original_width=width,
```

```
#       original_height=height,
#       xmin=xmin,
#       xmax=xmax,
#       ymin=ymin,
#       ymax=ymax,
#       target_width=244,
#       target_height=244
#     )

#     root.find('.//xmin').text = str(resized_xmin)
#     root.find('.//xmax').text = str(resized_xmax)
#     root.find('.//ymin').text = str(resized_ymin)
#     root.find('.//ymax').text = str(resized_ymax)
#     size_elem.find('width').text = str(desired_size[0])
#     size_elem.find('height').text = str(desired_size[1])

#     tree.write(xml_path)

#     img.save(image_path)
# for folder in folders:
#     # Get list of image files
#     image_files = [f for f in os.listdir(folder) if f.lower().endswith(('.png',
'.jpg', '.jpeg'))]

#     # Loop through each image file
#     for image_file in image_files:
#         image_path = os.path.join(folder, image_file)
#         xml_path = os.path.join(folder, image_file.replace('.jpg',
'.xml').replace('.jpeg', '.xml').replace('.png', '.xml'))

#         # Resize image and update XML
#         resize_image_and_xml(image_path, xml_path)

# print("Images and XML files have been resized and updated.")
```

۳-۲ . آموزش شبکه

در این فاز ابتدا از ویژگی های استفخراج شده از XML های متناظر هر عکس مقادیر استخراج کرده با مسیر تصاویر و خود عکس و Transform یک شی یکتا ساخته که برای آموزش آماده باشد سپس مدل faster RCNN را ایجاد میکنیم.

۱-۳-۲ . آماده سازی دیتا

```
import torch
from torch.utils.data import Dataset
import cv2
import pandas as pd
import numpy as np
from torch.utils.data import DataLoader

class_mapping = {'smoke': 1}
```

```
class WildFireSmoke_Dataset(Dataset):
    def __init__(self, image_path, categories, transforms=None):
```

```
        self.image_path = image_path
        self.categories = categories
        self.transforms = transforms
        self.df = self.load_data()
        self.images = self.df['filename'].unique()

        # Extract unique labels from the dataset
        xml_files = [f for f in os.listdir(self.image_path) if f.endswith('.xml')]
        unique_labels = set()

        for xml_file in xml_files:
            xml_path = os.path.join(self.image_path, xml_file)
            label = extract_labels(xml_path)
            unique_labels.add(label)

        self.unique_labels = list(unique_labels)

    def load_data(self):
        data  = {'filename': [], 'width': [], 'height': [], 'xmin': [], 'ymin': [],
'xmax': [], 'ymax': [], 'class': []}
        xml_files = [f for f in os.listdir(self.image_path) if f.endswith('.xml')]
        xml_files = [f for f in os.listdir(self.image_path) if f.endswith('.xml')]

        for xml_file in xml_files:
            xml_path = os.path.join(self.image_path, xml_file)
            tree = ET.parse(xml_path)
            root = tree.getroot()

            filename = root.find('.//filename').text
            width = int(root.find('.//width').text)
            height = int(root.find('.//height').text)

            for obj in root.findall('.//object'):
                class_name = obj.find('.//name').text
                bbox = obj.find('.//bndbox')
                xmin = int(bbox.find('.//xmin').text)
                ymin = int(bbox.find('.//ymin').text)
                xmax = int(bbox.find('.//xmax').text)
                ymax = int(bbox.find('.//ymax').text)

                data['filename'].append(filename)
                data['width'].append(width)
                data['height'].append(height)
                data['xmin'].append(xmin)
                data['ymin'].append(ymin)
                data['xmax'].append(xmax)
                data['ymax'].append(ymax)
                data['class'].append(1)

        df = pd.DataFrame(data)
        return df

        pass

    def __len__(self):
        return len(self.df)

    def __getitem__(self, idx):
```

```
        image_file = os.path.join(self.image_path,self.df.iloc[idx]['filename'])
        img = cv2.imread(image_file)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = img.astype(np.float32) / 255.0

        image_data = self.df[self.df['filename'] == self.df.iloc[idx]['filename']]
        xmins = image_data['xmin'].values
        ymins = image_data['ymin'].values
        xmaxs = image_data['xmax'].values
        ymaxs = image_data['ymax'].values

        boxes = torch.as_tensor(np.stack([xmins, ymins, xmaxs, ymaxs], axis=1),
dtype=torch.float32)
        labels = torch.as_tensor(image_data['class'].values, dtype=torch.int64)
        image_id = torch.tensor([idx])
        areas = (boxes[:,3] - boxes[:,1]) * (boxes[:,2] - boxes[:,0])
        areas = torch.as_tensor(areas, dtype=torch.float32)
        iscrowd = torch.zeros((len(labels),), dtype=torch.int64)

        target = {}
        target['boxes']    = boxes
        target['labels']   = labels
        target['image_id'] = image_id
        target['area']     = areas
        target['iscrowd']  = iscrowd

        if self.transforms is not
None:                                                                   #

                    #
            # Define
transformation
#
            transformed = self.transforms(image=img, bboxes=boxes,
labels=labels)                              #

                    #
            # Image
transformation
#
            img =
transformed['image']
  #

                    #
            # Target
transformation
#
            target['boxes'] = torch.as_tensor(transformed['bboxes'],
dtype=torch.float32)

        return torch.as_tensor(img, dtype=torch.float32), target

    def get_height_and_width(self, image):
        image_data = self.df.loc[self.df['filename'] == image]
        return image_data['width'].values[0], image_data['height'].values[0]
```

```
# Train set transformations
import albumentations as A
from albumentations.pytorch.transforms import ToTensorV2
transform_train = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.2),
    ToTensorV2(p=1)], bbox_params=A.BboxParams(format='pascal_voc',
label_fields=['labels']))

# Validation set transformations
transform_valid = A.Compose([
    ToTensorV2(p=1)], bbox_params=A.BboxParams(format='pascal_voc',
label_fields=['labels']))
```

```
def collate_fn(batch):
    return tuple(zip(*batch))
```

```
train_path = '/content/dataset/train'
valid_path = '/content/dataset/valid'
```

```
train_dataset = WildFireSmoke_Dataset(image_path=train_path, categories=['smoke'],
transforms=transform_train)
valid_dataset = WildFireSmoke_Dataset(image_path=valid_path, categories=['smoke'],
transforms=transform_valid)

# train_loader = DataLoader(train_dataset, batch_size=4, num_workers = 2,shuffle=True)
# valid_loader = DataLoader(train_dataset, batch_size=4,num_workers = 2, shuffle=False)

data_loader_train = torch.utils.data.DataLoader(
        train_dataset,
        batch_size  = 4,
        shuffle     = True,
        num_workers = 2,
        collate_fn  = collate_fn)

data_loader_valid = torch.utils.data.DataLoader(
        valid_dataset,
        batch_size  = 4,
        shuffle     = False,
        num_workers = 2,
        collate_fn  = collate_fn)
```

```
print(f"Length of train dataset: {len(train_dataset)}")
print(f"Length of valid dataset: {len(valid_dataset)}")
```

```
Length of train dataset: 516
Length of valid dataset: 147
```

```
train_dataset.df
```

	filename	width	height	xmin	ymin	xmax	ymax	class	
0	ck0u01dbyuoe30701p6pfzb0q_jpeg.rf.dc34aba65774...	640	480	159	167	422	321	1	

filename	width	height	xmin	ymin	xmax	ymax	class	
1	ck0kdox8w69td0838i6leuzy9_jpeg.rf.333d536f32a8...	640	480	124	215	191	314	1
2	ck0t4sh7pnk3v09447wbxifkk_jpeg.rf.d723fb8fb9b1...	640	480	445	228	583	254	1
3	ck0qb9do2fwc708385reizk6h_jpeg.rf.8353ce140548...	640	480	185	260	223	291	1
4	ck0kkg0u65u3q0863z6w2psqp_jpeg.rf.80b296c2869e...	640	480	468	193	541	245	1
...	...	...	...	...	...	...	...	...
511	ck0kfk23ukj1m0848e93um98b_jpeg.rf.457fd102e55b...	640	480	308	147	560	255	1
512	ck0tsyvp361ow084867dinue0_jpeg.rf.d47c7f4008f3...	640	480	499	246	637	288	1
513	ck0kkg80v5x900794hcgsb4e1_jpeg.rf.c14e4f981367...	640	480	121	202	194	313	1
514	ck0ndbtmr8oa50721hzgxpue8_jpeg.rf.98eed4485749...	640	480	310	159	443	252	1
515	ck0l8b13joiy30848vj3d2ndb_jpeg.rf.378c69f82ef9...	640	480	192	206	209	226	1

```
train_dataset[1]
```

```
(tensor([[[0.1961, 0.1961, 0.1961,  ..., 0.3922, 0.2000, 0.2314],
          [0.1961, 0.1961, 0.1961,  ..., 0.4039, 0.4549, 0.3098],
          [0.1961, 0.1961, 0.1961,  ..., 0.1098, 0.3216, 0.2392],
          ...,
          [0.2863, 0.2784, 0.2706,  ..., 0.2039, 0.1843, 0.1490],
          [0.2824, 0.2824, 0.2745,  ..., 0.2275, 0.2196, 0.2000],
          [0.2549, 0.2627, 0.2549,  ..., 0.1961, 0.2000, 0.2000]],

         [[0.2000, 0.2000, 0.2000,  ..., 0.3882, 0.1961, 0.2275],
          [0.2000, 0.2000, 0.2000,  ..., 0.4000, 0.4510, 0.3059],
          [0.2000, 0.2000, 0.2000,  ..., 0.1137, 0.3255, 0.2431],
          ...,
          [0.3216, 0.3137, 0.3059,  ..., 0.2118, 0.1922, 0.1569],
          [0.3176, 0.3176, 0.3098,  ..., 0.2353, 0.2275, 0.2078],
          [0.2902, 0.2980, 0.2902,  ..., 0.2039, 0.2078, 0.2078]],

         [[0.1451, 0.1451, 0.1451,  ..., 0.2706, 0.0784, 0.1098],
          [0.1451, 0.1451, 0.1451,  ..., 0.2784, 0.3294, 0.1843],
          [0.1451, 0.1451, 0.1451,  ..., 0.0000, 0.1922, 0.1098],
          ...,
          [0.3176, 0.3098, 0.3020,  ..., 0.1608, 0.1412, 0.1059],
          [0.3137, 0.3137, 0.3059,  ..., 0.1843, 0.1765, 0.1569],
          [0.2863, 0.2941, 0.2863,  ..., 0.1529, 0.1569, 0.1569]]]),
 {'boxes': tensor([[449., 215., 516., 314.]]),
  'labels': tensor([1]),
  'image_id': tensor([1]),
  'area': tensor([6633.]),
```

```
'iscrowd': tensor([0])})
```

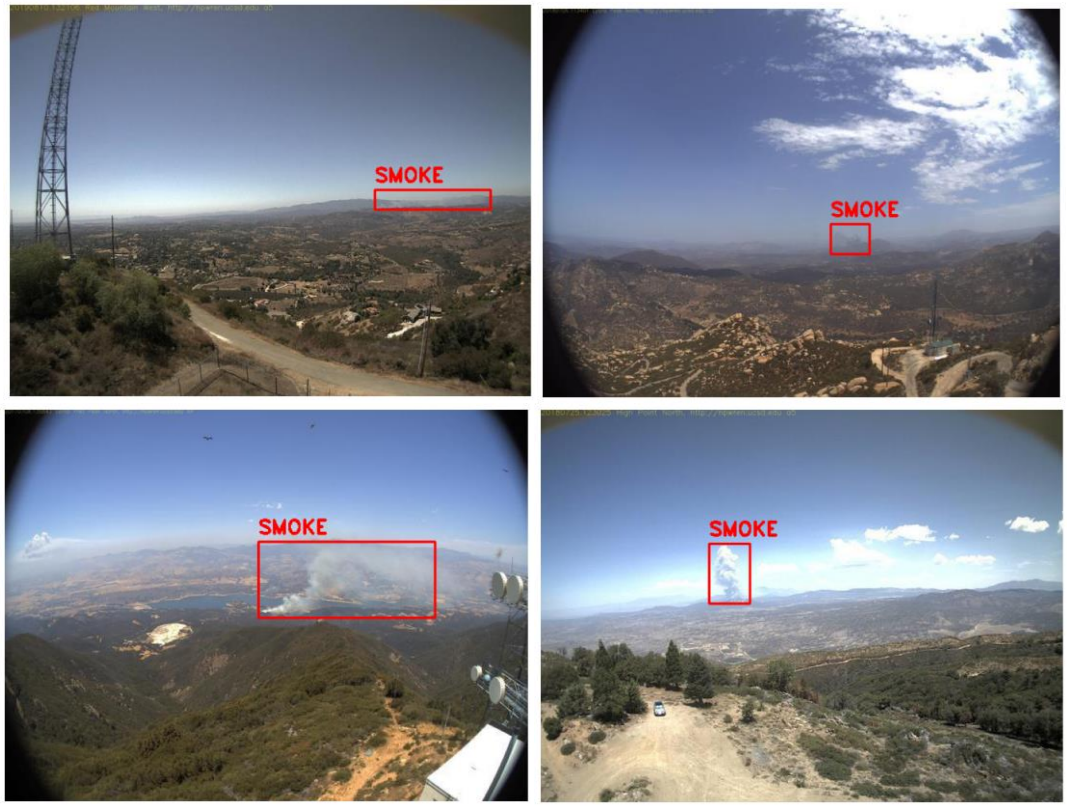
۲-۳-۲ . نمایش تصاویر با Labelهای مربوط از روی کلاس تشکیل شده‌ی بالا

```
def plot_images(images, targets):

    max_images = 4
    count_imgs = 0

    for image, target in zip(images, targets):

        if count_imgs == max_images:
            break;

        count_imgs += 1

        # Get the image and the boxes in the right format
        sample = image.permute(1,2,0).cpu().numpy()
        boxes  = target['boxes'].cpu().numpy().astype(np.int32)

        # Define the plot
        fig, ax = plt.subplots(1, 1, figsize=(10, 8))

        # Plot the boxes
        for box in boxes:
            cv2.rectangle(sample,
                          (box[0], box[1]),
                          (box[2], box[3]),
                          (1,0,0), 2);

            cv2.putText(sample, 'SMOKE',
                          (box[0], box[1]-10), cv2.FONT_HERSHEY_DUPLEX, 0.8, (1,0,0), 2)

        # Additional
        ax.set_axis_off();
        ax.imshow((sample * 255).astype(np.uint8))
# Train set - images and boxes
images, targets = next(iter(data_loader_train))
plot_images(images, targets)
```

شکل(2-7): نمایش تصاویر آموزش (4) مورد همراه **BBox**

همین کار را برای داده های Valid انجام می‌دهیم:

```
images, targets = next(iter(data_loader_valid))
plot_images(images, targets)
```



شکل(2-8): نصاویر **valid** همراه **BBox**

۲-۳-۳ . آموزش شبکه و پیاده سازی مدل

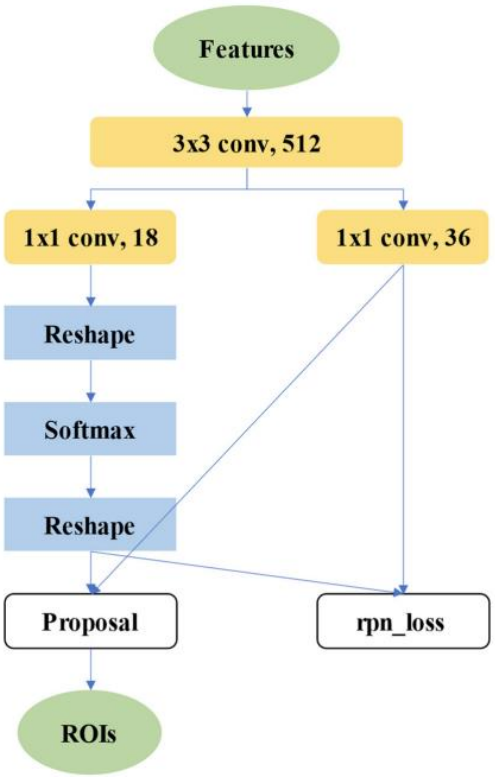
در ادامه مطابق مدل اارائه شده در مقاله میتوانیم مدل Faster RCNN را پیاده سازی کنیم که دو راه حل داریم. اول آنکه مدل را پیاده سازی و مقادیر مختلف را نیز مطابق آن در نظر بگیریم. دوم آنکه مدل را از کتابخانه import کرده وسر انتهای آن را تغییر بدهیم و مقادیر loss optimizer و.. متناظر را نیز تعریف کنیم(مدل را modify) کنیم ما در اینجا راه دوم را استفاده می‌کنیم.

مدل آموزش داده شده معمولا روی داده های COCO فراهم است

Available models, pre-trained on COCO dataset, at
https://pytorch.org/vision/master/models/faster_rcnn.html.

```
import torchvision
# Transfer learning: load a model pre-trained on COCO
detection_model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True)
num_classes = 2

# Get the number of input features for the classifier
in_features = detection_model.roi_heads.box_predictor.cls_score.in_features

# Replace the pre-trained head with a new one
detection_model.roi_heads.box_predictor =
torchvision.models.detection.faster_rcnn.FastRCNNPredictor(in_features, num_classes)
detection_model = detection_model.to(device)
```

اضافه کردن RPN دلخواه:



شکل (9-2): معماری **RPN** مقاله

```
rpn_input = backbone.output
rpn_layer = layers.Conv2D(256, (3, 3), padding='same', activation='relu')(rpn_input)
rpn_bbox = layers.Conv2D(4, (1, 1), activation='linear', name='rpn_bbox')(rpn_layer)
rpn_class = layers.Conv2D(2, (1, 1), activation='softmax', name='rpn_class')(rpn_layer)
```

اضافه کردن Poolingدلخواه

```
roi_pooling = layers.MaxPooling2D(pool_size=(7, 7))(rpn_class)

# Create the fully connected layers for object detection
flatten = layers.Flatten()(roi_pooling)
```

```
fc1 = layers.Dense(1024, activation='relu')(flatten)
fc2_class = layers.Dense(1, activation='sigmoid', name='fc2_class')(fc1)
fc2_bbox = layers.Dense(4, activation='linear', name='fc2_bbox')(fc1)
```

خب باز در ادامه میتوان از

```
from engine import train_one_epoch, evaluate
```

که فایل های آن در اینترنت موجود است برای ذخیره هنگام Train و Evaluate و... استفاده کرد برای همین آن را اضافه می‌کنیم و مدل ایجاد شده را بر اساس تغییرات در 20 تا epoch آموزش می‌دهیم

```
def training(model, train_loader, val_loader, epochs):

    # Optimizer and learning rate
    params = [p for p in model.parameters() if p.requires_grad]
    optimizer = torch.optim.SGD(params, lr=0.005, momentum=0.9, weight_decay=0.0005)

    # Adapter
    lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.1)

    for epoch in range(epochs):

        # Train for one epoch, printing every 'print_freq'
        train_one_epoch(model, optimizer, train_loader, device, epoch,
print_freq=len(data_loader_train))

        # Update the learning rate
        lr_scheduler.step()

        # Evaluate on the validation set
        evaluate(model, val_loader, device=device)

training(detection_model, data_loader_train, data_loader_valid, epochs=20)
```

```
Epoch: [0]  [  0/129]  eta: 0:07:16  lr: 0.000044  loss: 0.6764 (0.6764)
loss_classifier: 0.3909 (0.3909)  loss_box_reg: 0.0511 (0.0511)  loss_objectness: 0.2175
(0.2175)  loss_rpn_box_reg: 0.0168 (0.0168)  time: 3.3817  data: 0.2240  max mem: 4013
Epoch: [0]  [128/129]  eta: 0:00:01  lr: 0.005000  loss: 0.1422 (0.2154)
loss_classifier: 0.0468 (0.0828)  loss_box_reg: 0.0876 (0.0946)  loss_objectness: 0.0043
(0.0313)  loss_rpn_box_reg: 0.0044 (0.0068)  time: 1.0610  data: 0.0182  max mem: 4174
Epoch: [0] Total time: 0:02:18 (1.0745 s / it)
creating index...
index created!
Test:  [ 0/37]  eta: 0:00:25  model_time: 0.4866 (0.4866)  evaluator_time: 0.0071
(0.0071)  time: 0.7009  data: 0.1967  max mem: 4174
Test:  [36/37]  eta: 0:00:00  model_time: 0.4634 (0.4590)  evaluator_time: 0.0036
(0.0040)  time: 0.4830  data: 0.0153  max mem: 4174
Test: Total time: 0:00:18 (0.4924 s / it)
Averaged stats: model_time: 0.4634 (0.4590)  evaluator_time: 0.0036 (0.0040)
Accumulating evaluation results...
DONE (t=0.06s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.312
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.835
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.162
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.311
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.299
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.346
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.347
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.435
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.437
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.433
```

```
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.427
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.454
Epoch: [1]  [  0/129]  eta: 0:03:11  lr: 0.005000  loss: 0.1563 (0.1563)
loss_classifier: 0.0504 (0.0504)  loss_box_reg: 0.0921 (0.0921)  loss_objectness: 0.0078
(0.0078)  loss_rpn_box_reg: 0.0060 (0.0060)  time: 1.4810  data: 0.3846  max mem: 4174
...
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.579
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.454
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.557
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.675
```

همین طور که مشاهده میشود این فایل اضافه شده اطلاعات خوبی در هنگام هر step آموزش به ما می‌دهد که میتوانیم کیفیت مدل ساخته شده را بسنجیم.
مدل ساخته شده را ذخیره می‌کنیم:

```
torch.save(detection_model.state_dict(), '/content/drive/MyDrive/Wildfire Smoke
Detection/detection_model.pt')
detection_model.load_state_dict(torch.load('/content/drive/MyDrive/Wildfire Smoke
Detection/detection_model.pt'))
```

۴-۲ . بررسی داده های تست

برای بررسی روی داده های تست ابده باید loader مربوط ایجاد بکنیم

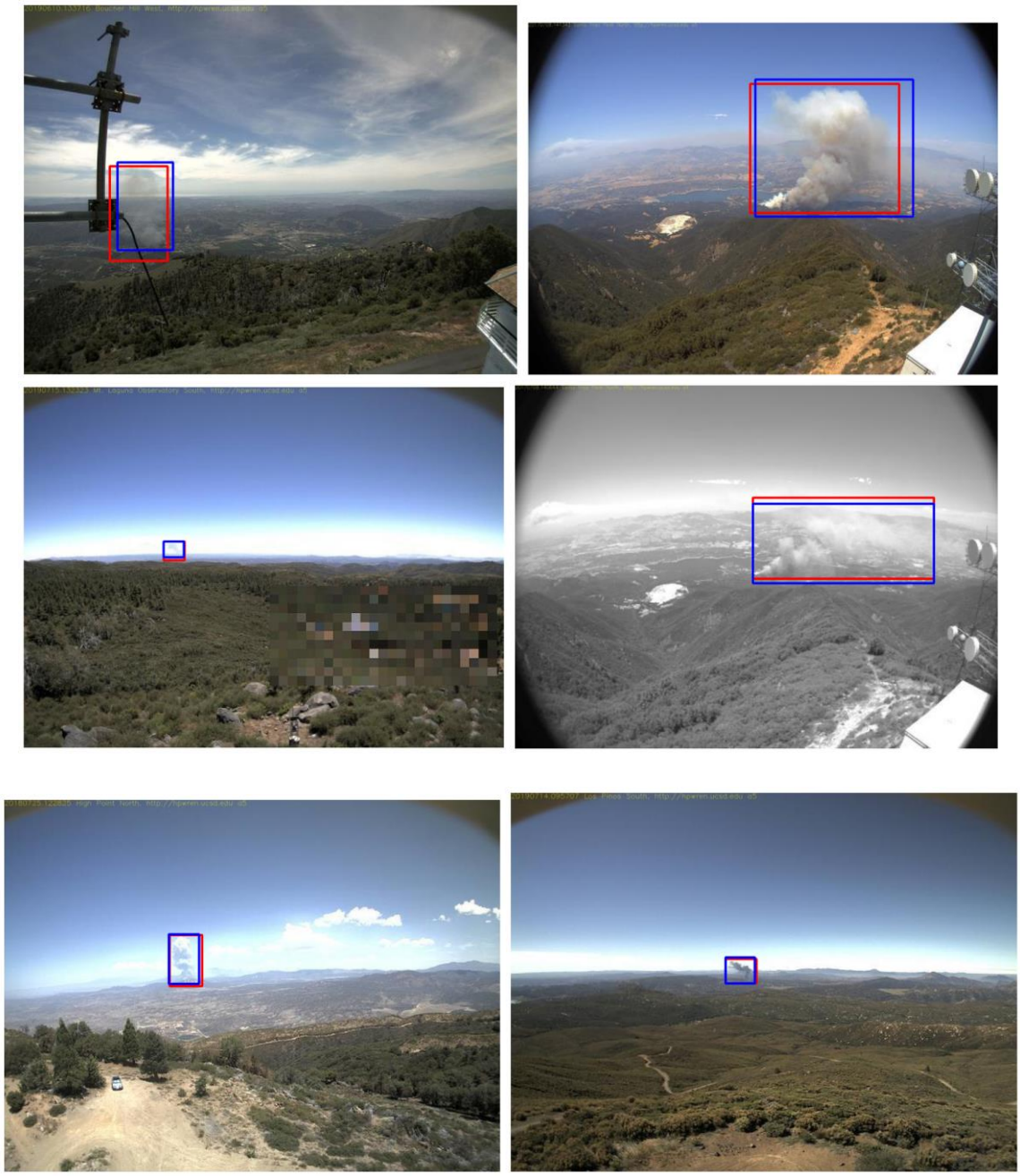
```
test_path = '/content/dataset/test'

# Validation set transformations
transform_test = A.Compose([
    ToTensorV2(p=1)], bbox_params=A.BboxParams(format='pascal_voc',
label_fields=['labels']))

test_dataset = WildFireSmoke_Dataset(image_path=test_path, categories=['smoke'],
transforms=transform_test)

data_loader_test = torch.utils.data.DataLoader(
        test_dataset,
        batch_size  = 4,
        shuffle     = True,
        num_workers = 2,
        collate_fn  = collate_fn)
```

سپس نمونه تصاویر خروجی را میگیریم که مدل برای تصاویر پیش بینی کرده است.
در اینجا label قرمز برای label original
و label آبی برای پیش بینی شده مدل است.

```
def view_sample(loader, model, device, threshold):

    images, targets = next(iter(loader))
    images = list(img.to(device) for img in images)
    targets = [{k: v.to(device) for k, v in t.items()} for t in targets]

    boxes_gt = targets[0]['boxes'].cpu().numpy().astype(np.int32)
    sample = images[0].permute(1,2,0).cpu().numpy()

    model.to(device)
    model.eval()
    cpu_device = torch.device('cpu')

    outputs = model(images)
    outputs = [{k: v.to(cpu_device) for k, v in t.items()} for t in outputs]
```

```
    fig, ax = plt.subplots(1, 1, figsize=(16, 8))

    # Paint ground_truth boxes (red)
    for box in boxes_gt:
        color_red = (1,0,0)
        cv2.rectangle(sample,
              (box[0], box[1]),
              (box[2], box[3]),
              color_red, 2)

    boxes = outputs[0]['boxes'].data.cpu().numpy().astype(np.int32)
    scores = outputs[0]['scores'].data.cpu().numpy()

    pred_box_count = 0
    for box, p in zip(boxes, scores):
        if p > threshold:
            pred_box_count += 1
            color_blue = (0,0,1)
            cv2.rectangle(sample,
                  (box[0], box[1]),
                  (box[2], box[3]),
                  color_blue, 2)

    print(f'Predicted {pred_box_count} BBoxes (blue); Number of GT BBoxes (red) :
{len(boxes_gt)}')
    ax.set_axis_off()
    ax.imshow((sample * 255).astype(np.uint8))
```

```
for i in range(10):
    view_sample(data_loader_test, detection_model, device, 0.6)
```



شکل (10-2): نمونه تصاویر پیش بینی **BBox** توسط مدل

همان طور که مشاهده می‌شود مدل به خوبی پیش بینی‌های درستی نظیر قرمز از خود نشان می‌دهد و نتایج با دقت بالا نزدیک اندازه نزدیک است.

ضعف مدل:

مدل اگر چه نتایج درستی نشان داده اما در اندکی از تصاویر خطا داشته مشابه:

شکل (11-2): نمونه تصویر با **BBox** اشتباه

همان طور که مشاهده می‌شود مدل دو تا Bounding Box برای این تصویر در نظر گرفته است که میتوان با افزایش داده ها / epochها و اندازه گیری مقادیر گوناگون یا استفاده از مدل های پیشرفته تر این موضوع را بر طرف ساخت تا دقت مدل نیز بالاتر برود. همچنین میتوان از روش های متنوع تر برای استفاده از Custom Back Bone مدل استفاده کرد.