Moving Object Detection & Classification

Inspired by “A Lightweight Gaussian-Based Model for Fast Detection and Classification of Moving Objects”

A peek into the data

Describing the data

- The data set is made up of multiple images taken from the dashboard camera (first-person view) of cars while they were in motion, and each image is labelled multiple times, each label being a moving object that can be seen in that image.

The training examples are composed of the image ID, the dimensions of the bounding box of the moving object in that image, and the class of that object.

There are 5 different classes in the dataset, each corresponding to an ID.

1: 'car',

2: 'truck',

3: 'pedestrian',

4: 'bicyclist',

5: 'light'

```
[10] df.head()
```

	frame	xmin	xmax	ymin	ymax	class_id
0	1478019952686311006.jpg	237	251	143	155	1
1	1478019952686311006.jpg	437	454	120	186	3
2	1478019953180167674.jpg	218	231	146	158	1
3	1478019953689774621.jpg	171	182	141	154	2
4	1478019953689774621.jpg	179	191	144	155	1

We can notice that the data has multiple training examples for each image, with each row showing the bounding box of a single object in an image and it's class.

In this example there's 5 different moving objects in the image: a truck and 4 cars.

```
[17] img_id = '1478019953689774621.jpg'
     img_details = df[df['frame']==img_id]
     img_details
```

	frame	xmin	xmax	ymin	ymax	class_id
3	1478019953689774621.jpg	171	182	141	154	2
4	1478019953689774621.jpg	179	191	144	155	1
5	1478019953689774621.jpg	206	220	145	156	1
6	1478019953689774621.jpg	385	420	122	152	1
7	1478019953689774621.jpg	411	462	124	148	1

As we can see, there are a total of 165k training examples in the dataset.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 165105 entries, 0 to 165104
Data columns (total 6 columns):
 #   Column    Non-Null Count   Dtype 
---  ------    --------------   ----- 
 0   frame     165105 non-null  object
 1   xmin      165105 non-null  int64 
 2   xmax      165105 non-null  int64 
 3   ymin      165105 non-null  int64 
 4   ymax      165105 non-null  int64 
 5   class_id  165105 non-null  int64 
dtypes: int64(5), object(1)
memory usage: 7.6+ MB
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 165105 entries, 0 to 165104
Data columns (total 6 columns):
 #   Column    Non-Null Count   Dtype 
---  ------    --------------   ----- 
 0   frame     165105 non-null  object
 1   xmin      165105 non-null  int64 
 2   xmax      165105 non-null  int64 
 3   ymin      165105 non-null  int64 
 4   ymax      165105 non-null  int64 
 5   class_id  165105 non-null  int64 
dtypes: int64(5), object(1)
memory usage: 7.6+ MB
```

```
[25] labels = {1:'car',
               2:'truck',
               3:'pedestrian',
               4:'bicyclist',
               5:'light'}
     target2labels = labels.copy()
     target2labels
```

```
{1: 'car', 2: 'truck', 3: 'pedestrian', 4: 'bicyclist', 5: 'light'}
```

```
[26] class_counts = df['class_id'].value_counts(sort=True).to_dict()
     class_counts = dict(sorted(class_counts.items()))
     class_counts
```

```
{1: 123314, 2: 7322, 3: 15540, 4: 1676, 5: 17253}
```

Number of images for each class

```python
def new_df(df):
    for (i, fname) in a:
        fpath = f'/content/files/images/{fname}'
        img = np.asarray(Image.open(fpath))
        h ,w ,_ = img.shape
        df.iloc[i, 1] /= w  # xmin
        df.iloc[i, 2] /= w  # xmax
        df.iloc[i, 3] /= h  # ymin
        df.iloc[i, 4] /= h  # ymax
    return df
df1 = new_df(df)
df1.head()
```

	frame	xmin	xmax	ymin	ymax	class_id
0	1478019952686311006.jpg	0.493750	0.522917	0.476667	0.516667	1
1	1478019952686311006.jpg	0.910417	0.945833	0.400000	0.620000	3
2	1478019953180167674.jpg	0.454167	0.481250	0.486667	0.526667	1
3	1478019953689774621.jpg	0.356250	0.379167	0.470000	0.513333	2
4	1478019953689774621.jpg	0.372917	0.397917	0.480000	0.516667	1

Rescale the boundary-box size

```
%load /content/drive/MyDrive/TRG-net-master/trgnet/backbones/components.py
%load /content/drive/MyDrive/TRG-net-master/trgnet/backbones/fpn.py
%load /content/drive/MyDrive/TRG-net-master/trgnet/backbones/mobilenetv3.py
%load /content/drive/MyDrive/TRG-net-master/trgnet/backbones/utils.py

%load /content/drive/MyDrive/TRG-net-master/trgnet/training/reference/coco_eval.py
%load /content/drive/MyDrive/TRG-net-master/trgnet/training/reference/coco_utils.py
%load /content/drive/MyDrive/TRG-net-master/trgnet/training/reference/engine.py
%load /content/drive/MyDrive/TRG-net-master/trgnet/training/reference/transforms.py
%load /content/drive/MyDrive/TRG-net-master/trgnet/training/reference/utils.py

%load /content/drive/MyDrive/TRG-net-master/trgnet/training/train.py
%load /content/drive/MyDrive/TRG-net-master/trgnet/training/utils.py

%load /content/drive/MyDrive/TRG-net-master/trgnet/anchor.py
%load /content/drive/MyDrive/TRG-net-master/trgnet/data.py
%load /content/drive/MyDrive/TRG-net-master/trgnet/grpm.py
%load /content/drive/MyDrive/TRG-net-master/trgnet/misc.py
%load /content/drive/MyDrive/TRG-net-master/trgnet/roi_heads.py
%load /content/drive/MyDrive/TRG-net-master/trgnet/rpn.py
%load /content/drive/MyDrive/TRG-net-master/trgnet/timer.py
%load /content/drive/MyDrive/TRG-net-master/trgnet/trg.py
%load /content/drive/MyDrive/TRG-net-master/trgnet/utils.py
%load /content/drive/MyDrive/TRG-net-master/trgnet/zoo.py

%load /content/drive/MyDrive/TRG-net-master/setup.py
```

Loading the model

```python
import time

import cv2
import torch
import torchvision
import torchvision.transforms as transforms


from PIL import Image

from trgnet.zoo import trgnet_mobilenet_v3_large
```

Import needed libraries: torch, open-cv, PIL, trgnet

```python
class SelfDrivingCarDataset(Dataset):
  w , h = 224 , 224
  def __init__(self , df  , image_root_dir = '/content/files/images' ):
    self.image_dir = image_root_dir
    self.df = df
    self.files = glob.glob(self.image_dir + '/*.jpg')
    self.image_infos = df.frame.unique()

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

  def __getitem__(self ,ix ):
    img_id = self.image_infos[ix]
    img_path = f'/content/files/images/{img_id}'
    img = Image.open(img_path).convert('RGB')
    img = np.array(img.resize((self.w ,self.h) , resample = Image.BILINEAR))/255.
    data = df[df['frame'] == img_id]
    labels = data['class_id'].values.tolist()
    data = data[['xmin', 'ymin' ,'xmax' ,'ymax']].values
    data[:,[0,2]] *=  self.w
    data[:,[1,3]] *=  self.h
    boxes = data.astype(np.uint32).tolist()
    target = {}
    target["boxes"] = torch.Tensor(boxes).float()
    target["labels"] = torch.Tensor([i for i in labels]).long()
    img = preprocess_image(img)
    return img , target

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

Since a single image appears multiple times with different bounding box and label values, we will group all those values together in a single data structure, which will contain the image, the list of bounding boxes, and the list of labels.

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

from sklearn.model_selection import train_test_split as tts

x, test_ids = tts(df1.frame.unique(), test_size = 0.15, random_state = 99)  # test size will be 15%
trn_ids, val_ids = tts(x, train_size = 0.8225, random_state = 99)  # train size will be 70%, val 15%

trn_df, val_df, test_df = df1[df1['frame'].isin(trn_ids)], df1[df1['frame'].isin(val_ids)], df1[df1['frame'].isin(test_ids)]

train_ds = SelfDrivingCarDataset(trn_df)
val_ds = SelfDrivingCarDataset(val_df)
test_ds = SelfDrivingCarDataset(test_df)

train_loader = DataLoader(train_ds,  batch_size = 4, collate_fn = train_ds.collate_fn, drop_last =True)
val_loader = DataLoader(val_ds, batch_size = 4, collate_fn = val_ds.collate_fn, drop_last = True)
test_loader = DataLoader(test_ds, batch_size = 4, collate_fn = test_ds.collate_fn, drop_last = True)
```

Obtaining the unique frames and splitting them into training, validation, and testing

Number of images in (taining ,validation,testing)

```
len(train_ds), len(val_ds), len(test_ds)
```

```
(2690, 581, 578)
```

Initializing the model parameter

```
model = trgnet_mobilenet_v3_large(
    pretrained=False, grpm_min_area=50, grpm_lr=0.01,
    grpm_show_output=True,num_classes=5).to(device)
```

Creating functions to train, validate, and test in batches.

The training function will use model.train(), calculate the losses, then calculate a gradient step using loss.backward() and optimizer.step()

```python
def train_batch(inputs, model , optimizer):
  model.train()
  input , targets = inputs
  input = list(image.to(device) for image in input)
  targets = [{k:v.to(device) for k,v in t.items()} for t in targets]
  optimizer.zero_grad()
  losses = model(input, targets)
  loss = sum(loss for loss in losses.values())
  loss.backward()
  optimizer.step()
  return loss, losses

@torch.no_grad()
def validate_batch(inputs, models):
  input, targets = inputs
  input = list(image.to(device) for image in input)
  targets = [{k:v.to(device) for k,v in t.items()} for t in targets]
  losses = model(input, targets)
  loss = sum(loss for loss in losses.values())
  return loss, losses

@torch.no_grad()
def test_batch(inputs, models):
  input, targets = inputs
  input = list(image.to(device) for image in input)
  targets = [{k:v.to(device) for k,v in t.items()} for t in targets]
  losses = model(input, targets)
  loss = sum(loss for loss in losses.values())
  return loss, losses
```

We use the stochastic gradient descent optimizer which include hyper parameter (learning rate ,momentum,weight_decay)

The classifier and box regression loss are the output of the final predictor at the end of model, while the object and the rpn_box_reg losses are related to the GRPM model and how well it can produces the proposals.

```
optimizer = torch.optim.SGD(model.parameters() , lr=0.005 ,
                            momentum = 0.9 , weight_decay= 0.0005 )

loss_criterions = ['loss_classifier',
          'loss_box_reg',
          'loss_objectness',
          'loss_rpn_box_reg']

EPOCHS = 5
```