https://github.com/jhcho99 / gsrtr:

The code is modified and adapted from these amazing repositories:

* [End-to-End Object Detection with Transformers](https://github.com/facebookresearch/detr)
* [Grounded Situation Recognition](https://github.com/allenai/swig) (Allen Institute for Artificial Intelligence)

[*https://prior.allenai.org/projects/gsr*](https://prior.allenai.org/projects/gsr).

SWiG benchmark.

<https://github.com/allenai/swig>

## [gsrtr](https://github.com/jhcho99/gsrtr)/[datasets](https://github.com/jhcho99/gsrtr/tree/master/datasets)/****swig.py****

## https://github.com/jhcho99/gsrtr/blob/master/SWiG/SWiG\_jsons/train\_classes.csv

train\_classes.csv : for each role : 3 objets chosen with sequence number

>>>> can be used for SSL as similar to previous recognized object

Example:

"n00017222": {

"gloss": ["plant", "flora", "plant life"],

"def": "(botany) a living organism lacking the power of locomotion"

}

https://github.com/jhcho99/CoFormer.

<https://github.com/kellyiss/SituFormer>

|  |  |
| --- | --- |
| https://github.com/jhcho99/gsrtr/blob/master/datasets/swig.py | |
|  | <https://pytorch.org/docs/stable/multiprocessing.html>:  torch.multiprocessing is a wrapper around the native [multiprocessing](https://docs.python.org/3/library/multiprocessing.html#module-multiprocessing) module. It registers custom reducers, that use shared memory to provide shared views on the same data in different processes. Once the tensor/storage is moved to shared\_memory. it will be possible to send it to other processes without making any copies.  It’s enough to change **import multiprocessing** to **import torch.multiprocessing** to have all the tensors sent through the queues or shared via other mechanisms, moved to shared memory.  **Strategy management:**  torch.multiprocessing.get\_all\_sharing\_strategies()  Returns a set of sharing strategies supported on a current system.  torch.multiprocessing.get\_sharing\_strategy()  Returns the current strategy for sharing CPU tensors.  torch.multiprocessing.set\_sharing\_strategy(new\_strategy)  Sets the strategy for sharing CPU tensors.  Parameters  new\_strategy (str) – Name of the selected strategy. Should be one of the values returned by get\_all\_sharing\_strategies(). |
|  | from torchvision import transforms  L51: self.color\_change = transforms.Compose([transforms.ColorJitter(hue=0.1, saturation=0.1, brightness=0.1), transforms.RandomGrayscale(p=0.3)]) |
| import datasets.transforms as T  L115>>>L144 in https://github.com/facebookresearch/detr/blob/main/datasets/coco.py |
|  | Csv files :  <https://docs.python.org/fr/3.6/library/csv.html>  with open(self.class\_list, 'r') as file:  f= csv.reader(file, delimiter=',') |
|  | self.load\_classes train.classes.csv  L53,L54  self.classes, self.idx\_to\_class = self.load\_classes(csv.reader(file, delimiter=',')) |
|  | .json files: , [train.json](https://github.com/jhcho99/gsrtr/blob/master/SWiG/SWiG_jsons/train.json), [dev.json](https://github.com/jhcho99/gsrtr/blob/master/SWiG/SWiG_jsons/dev.json), [test.json](https://github.com/jhcho99/gsrtr/blob/master/SWiG/SWiG_jsons/test.json) [imsitu\_space.json](https://github.com/jhcho99/gsrtr/blob/master/SWiG/SWiG_jsons/imsitu_space.json)  L56,L57  Json to txt   |  | | --- | | With open(self.ann\_file) as file:  self.SWiG\_json = json.load(file) | |
|  | [train.json](https://github.com/allenai/swig/blob/master/SWiG_jsons/train.json),  verb\_orders= imsitu\_space.json [‘verbs’],  [train\_classes.csv](https://github.com/jhcho99/gsrtr/blob/master/SWiG/SWiG_jsons/train_classes.csv)[classes, idx\_to\_class].  self.\_read\_annotations:  self.SWiG\_json, verb\_info, self.classes  load data respectively from [train.json](https://github.com/allenai/swig/blob/master/SWiG_jsons/train.json), verb\_info=verb\_orders= imsitu\_space.json [“**verbs**"], [train\_classes.csv](https://github.com/jhcho99/gsrtr/blob/master/SWiG/SWiG_jsons/train_classes.csv)[classes, idx\_to\_class].  imsitu\_space.json:  imsitu\_space.json {"**verbs**"::{verb :{ "framenet": , def : , abstract : , "order": , roles : }}  "nouns" :{e\_i : { "gloss": , def : } , e\_j,.. }  The same image may have 6 annotations  while total\_anns < 6: # |
|  | img\_file = f"{self.img\_folder}/" + image  result[img\_file].append({'x1': x1, 'x2': x2, 'y1': y1, 'y2': y2, 'class1': class1, 'class2': class2, 'class3': class3})  [all\_ims.txt](https://github.com/allenai/swig/blob/master/all_ims.txt)  self.image\_data = self.\_read\_annotations(self.SWiG\_json, verb\_info, self.classes)  self.image\_names = list(self.image\_data**.keys())** |
|  | L62 to L65  verb\_path file : [verb\_indices.txt](https://github.com/jhcho99/gsrtr/blob/master/SWiG/SWiG_jsons/verb_indices.txt)  self.load\_verb  role\_path file : [role\_indices.txt](https://github.com/jhcho99/gsrtr/blob/master/SWiG/SWiG_jsons/role_indices.txt) |
|  | Try to index all things (images, roles, classes,verbs,etc)  self.image\_to\_image\_idx = {}  Dict : for each image, we associate an idx.  L67>>L71  L67>>L71 : for indexing images |
|  | verb\_info=verb\_orders= imsitu\_space.json [“**verbs**"],  for verb, value in verb\_info.items()}:  Example of item of verb\_info  "tattooing": {"framenet": "Create\_physical\_artwork", "abstract": "AGENT tattooed TARGET with TOOL in PLACE", "def": "to mark the skin with permanent colors and patterns", "**order":** ["agent", "target", "tool", "place"], "roles": {"tool": {"framenet": "instrument", "def": "The tool used"}, "place": {"framenet": "place", "def": "The location where the tattoo event is happening"}, "target": {"framenet": "representation", "def": "The entity being tattooed"}, "agent": {"framenet": "creator", "def": "The entity doing the tattoo action"}}}  # verb\_role  self.verb\_role = {verb: value['order'] for verb, value in verb\_info.items()}  # for each verb, the indices of roles in the frame.  self.vidx\_ridx = [[self.role\_to\_idx[role] for role in self.verb\_role[verb]] for verb in self.idx\_to\_verb]  self.verb\_role : associate each verb with each roles :  verb in the list of “verbs” =verb\_info.items(1)  value in the list of nouns= verb\_info.items(2) |
|  | self.verb\_role = {verb: value['order'] for verb, value in verb\_info.items()} |
|  | def load\_image(self, image\_index):  im = Image.open(self.image\_names[image\_index])  im = im.convert('RGB')  if self.is\_training:  im = np.array(self.color\_change(im))  else:  im = np.array(im)  return im.astype(np.float32) / 255.0 |
|  | def load\_annotations(self, image\_index):  # get ground truth annotations  annotation\_list = self.image\_data[self.image\_names[image\_index]]  annotations = np.ones((0, 7)) \* -1  # some images appear to miss annotations (like image with id 257034)  if len(annotation\_list) == 0:  return annotations  # parse annotations  for idx, a in enumerate(annotation\_list):  annotation = np.ones((1, 7)) \* -1 # allow for 3 annotations  annotation[0, 0] = a['x1']  annotation[0, 1] = a['y1']  annotation[0, 2] = a['x2']  annotation[0, 3] = a['y2']  annotation[0, 4] = self.name\_to\_label(a['class1'])  annotation[0, 5] = self.name\_to\_label(a['class2'])  annotation[0, 6] = self.name\_to\_label(a['class3'])  annotations = np.append(annotations, annotation, axis=0)  return annotations |
|  | import **skimage.transform**  from torchvision import **transforms**, utils  def \_\_init\_\_(self, img\_folder, ann\_file, class\_list, verb\_path, role\_path, verb\_info, is\_training, **transform=None**):  **self.transform = transform**  def \_\_getitem\_\_(self, idx):  ……  **if self.transform:**  **sample = self.transform(sample)**  **return sample** |
|  | #image and annotation loading, make the samples in json format  def \_\_getitem\_\_(self, idx):  img = self.load\_image(idx)  annot = self.load\_annotations(idx)  verb = self.image\_names[idx].split('/')[2]  verb = verb.split('\_')[0]  verb\_idx = self.verb\_to\_idx[verb]  verb\_role = self.verb\_info[verb]['order']  verb\_role\_idx = [self.role\_to\_idx[role] for role in verb\_role]  sample = {'img': img, 'annot': annot, 'img\_name': self.image\_names[idx], 'verb\_idx': verb\_idx, 'verb\_role\_idx': verb\_role\_idx}  if self.transform:  sample = self.transform(sample)  return sample |
|  | def name\_to\_label(self, name):  return self.classes[name]  def num\_nouns(self):  return max(self.classes.values()) + 1  def image\_aspect\_ratio(self, image\_index):  image = Image.open(self.image\_names[image\_index])  return float(image.width) / float(image.height) |
|  | #Data batching  def collater(data): |
|  | Shift of the image in the predefinced (Batch size, max\_height,max\_width) tensor  shift\_0  shift\_1 |
|  | **util.misc**  L271: util.misc.nested\_tensor\_from\_tensor\_list ~padding : the list of tensors(padded\_imgs) in one tensor |
|  | **util.box\_ops**  'boxes': util.box\_ops.swig\_box\_xyxy\_to\_cxcywh(annot[:, :4], mw, mh, gt=True),  def box\_xyxy\_to\_cxcywh(x):  x0, y0, x1, y1 = x.unbind(-1)  b = [(x0 + x1) / 2, (y0 + y1) / 2,  (x1 - x0), (y1 - y0)]  return torch.stack(b, dim=-1) |
|  | |  | | --- | | L271>>>>L284  return(util.misc.nested\_tensor\_from\_tensor\_list(padded\_imgs), | |  | [{'verbs': vi, | |  | 'roles': vri, | |  | 'boxes': util.box\_ops.swig\_box\_xyxy\_to\_cxcywh(annot[:, :4], mw, mh, gt=True), | |  | 'labels': annot[:, -3:], | |  | 'width': w, | |  | 'height': h, | |  | 'shift\_0': s0, | |  | 'shift\_1': s1, | |  | 'scale': sc, | |  | 'max\_width': mw, | |  | 'max\_height': mh, | |  | 'img\_name': im} | |  |  | |  | for vi, vri, annot, w, h, s0, s1, sc, im in zip(verb\_indices, verb\_role\_indices, annot\_padded, widths, heights, shift\_0, shift\_1, scales, img\_names)] ) | |  |  | |  |  | |
|  | **class Resizer(object):**  **def \_\_init\_\_(self, is\_for\_training):**  self.is\_for\_training = is\_for\_training  **def \_\_call\_\_(self, sample, min\_side=300, max\_side=400):**  image, annots, image\_name = **sample['img'], sample['annot'], sample['img\_name']**  return {**'img': torch.from\_numpy(new\_image), 'annot': torch.from\_numpy(annots), 'scale': scale, 'img\_name': image\_name, 'verb\_idx': sample['verb\_idx'], 'verb\_role\_idx': sample['verb\_role\_idx'], 'shift\_1': shift\_1, 'shift\_0': shift\_0**} |
|  | class Resizer(object): L287----->L329 scale, scale\_factor  #resize the image with the computed scale  image = skimage.transform.resize(image, (int(round(rows\_orig \* scale)), int(round((cols\_orig \* scale)))))  rows, cols, cns = image.shape |
|  | new\_image = np.zeros((rows, cols, cns)).astype(np.float32)  new\_image[:rows, :cols, :] = image.astype(np.float32)  shift\_1 = int((400 - cols) \* 0.5)  shift\_0 = int((400 - rows) \* 0.5)  != -1 :Not equal  \*= : times  annots[:, 0][annots[:, 0] != -1] = annots[:, 0][annots[:, 0] != -1] + shift\_1  annots[:, 1][annots[:, 1] != -1] = annots[:, 1][annots[:, 1] != -1] + shift\_  annots[:, 2][annots[:, 2] != -1] = annots[:, 2][annots[:, 2] != -1] + shift\_1  annots[:, 3][annots[:, 3] != -1] = annots[:, 3][annots[:, 3] != -1] + shift\_0  annotation :numpy >> annotation = np.ones((1, 7)) \* -1  image : numpy (PIL, )  return {'img': torch.from\_numpy(new\_image), 'annot': torch.from\_numpy(annots), 'scale': scale, 'img\_name': image\_name, 'verb\_idx': sample['verb\_idx'], 'verb\_role\_idx': sample['verb\_role\_idx'], 'shift\_1': shift\_1, 'shift\_0': shift\_0} |
|  | **class Augmenter(object):**  def \_\_call\_\_(self, sample, flip\_x=0.5):  image, annots, img\_name = sample['img'], sample['annot'], sample['img\_name']  **sample = {'img': image, 'annot': annots, 'img\_name': img\_name, 'verb\_idx': sample['verb\_idx'], 'verb\_role\_idx':**  **sample['verb\_role\_idx']}**  **return sample** |
|  | class Augmenter(object):   |  | | --- | |  | | If np.random.rand() < flip\_x: | |  | image = image[:, ::-1, :] | |  | rows, cols, channels = image.shape | |  |  | |  | x1 = annots[:, 0].copy() | |  | x2 = annots[:, 2].copy() | |  |  | |  | annots[:, 0][annots[:, 0] != -1] = cols - x2[annots[:, 0] != -1] | |  | annots[:, 2][annots[:, 2] != -1] = cols - x1[annots[:, 2] != -1] | |  |  | |
|  | **class Normalizer(object):**  def \_\_init\_\_(self):  self.mean = np.array([[[0.485, 0.456, 0.406]]])  self.std = np.array([[[0.229, 0.224, 0.225]]])  def \_\_call\_\_(self, **sample**):  image, annots = sample['img'], sample['annot']  **return {'img': ((image.astype(np.float32) - self.mean) / self.std), 'annot': annots, 'img\_name': sample['img\_name'],**  **'verb\_idx': sample['verb\_idx'], 'verb\_role\_idx': sample['verb\_role\_idx']}** |
|  | class Normalizer(object) (-mean/std)  self.mean = np.array([[[0.485, 0.456, 0.406]]])  self.std = np.array([[[0.229, 0.224, 0.225]]])  'img': ((image.astype(np.float32) - self.mean) / self.std) |
|  | **class UnNormalizer(object):**  def \_\_init\_\_(self, mean=None, std=None):  if mean is None:  self.mean = [0.485, 0.456, 0.406]  else:  self.mean = mean  if std is None:  self.std = [0.229, 0.224, 0.225]  else:  self.std = std  **def \_\_call\_\_(self, tensor):**  **for t, m, s in zip(tensor, self.mean, self.std):**  **t.mul\_(s).add\_(m)**  **return tensor** |
|  | class UnNormalizer(object):  for t, m, s in zip(tensor, self.mean, self.std):  t.mul\_(s).add\_(m) torch tensor |
|  | **def build(image\_set, args):**  **root = Path(args.swig\_path)**  **img\_folder = root / "images\_512"**  **PATHS = {**  **"train": root / "SWiG\_jsons" / "train.json",**  **"val": root / "SWiG\_jsons" / "dev.json",**  **"test": root / "SWiG\_jsons" / "test.json",**  **}**  **ann\_file = PATHS[image\_set]**  **classes\_file = Path(args.swig\_path) / "SWiG\_jsons" / "train\_classes.csv"**  **verb\_path = Path(args.swig\_path) / "SWiG\_jsons" / "verb\_indices.txt"**  **role\_path = Path(args.swig\_path) / "SWiG\_jsons" / "role\_indices.txt"**  **with open(f'{args.swig\_path}/SWiG\_jsons/imsitu\_space.json') as f:**  **all = json.load(f)**  **verb\_orders = all['verbs']**  **is\_training = (image\_set == 'train')**  **TRANSFORMS = {**  **"train": transforms.Compose([Normalizer(), Augmenter(), Resizer(True)]),**  **"val": transforms.Compose([Normalizer(), Resizer(False)]),**  **"test": transforms.Compose([Normalizer(), Resizer(False)]),**  **}**  **tfs = TRANSFORMS[image\_set]**    **dataset = CSVDataset(img\_folder=str(img\_folder),**  **ann\_file=ann\_file,**  **class\_list=classes\_file,**  **verb\_path=verb\_path,**  **role\_path=role\_path,**  **verb\_info=verb\_orders,**  **is\_training=is\_training,**  **transform=tfs)**  **if is\_training:**  **args.SWiG\_json\_train = dataset.SWiG\_json**  **else:**  **if not args.test:**  **args.SWiG\_json\_dev = dataset.SWiG\_json**  **else:**  **args.SWiG\_json\_test = dataset.SWiG\_json**  **args.idx\_to\_verb = dataset.idx\_to\_verb**  **args.idx\_to\_role = dataset.idx\_to\_role**  **args.idx\_to\_class = dataset.idx\_to\_class**  **args.vidx\_ridx = dataset.vidx\_ridx**  **return dataset** |