Reproducibility Companion Paper: Visual Relation of Interest Detection

Fan Yu[1,2], Haonan Wang[1], Tongwei Ren[1,2,*], Jinhui Tang[3], Gangshan Wu[1]
Jingjing Chen[4], Zhenzhong Kuang[5]
[1] State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
[2] Shenzhen Research Institute of Nanjing University, Shenzhen, China
[3] School of Computer Science, Nanjing University of Science and Technology, Nanjing, China
[4] Shanghai Key Lab of Intelligent Information Processing, School of Computer Science, Fudan University, Shanghai, China
[5] Key Laboratory of Complex Systems Modeling and Simulation, School of Computer Science and Technology, Hangzhou Dianzi University
{yf,haonan.wang}@smail.nju.edu.cn,rentw@nju.edu.cn,jinhuitang@njust.edu.cn,gswu@nju.edu.cn
chenjingjing.tju@gmail.com,zzkuang@hdu.edu.cn

ABSTRACT

In this companion paper, we provide the details of the reproducibility artifacts of the paper "Visual Relation of Interest Detection" presented at MM'20. Visual Relation of Interest Detection (VROID) aims to detect visual relations that are important for conveying the main content of an image. In this paper, we explain the file structure of the source code and publish the details of our ViROI dataset, which can be used to retrain the model with custom parameters. We also detail the scripts for component analysis and comparison with other methods and list the parameters that can be modified for custom training and inference.

CCS CONCEPTS

• **Computing methodologies** → **Computer vision**.

KEYWORDS

Visual relation of interest; visual relation detection; interest propagation network; interest estimation

1 ARTIFACTS DESCRIPTION

1.1 Introduction

We proposed a novel task named Visual Relation of Interest Detection (VROID) [15] at MM'20 to extract visual relations that are more semantically important. VROID evolves from the tasks of Visual Relation Detection (VRD) [5] and Instance of Interest Detection (IOID) task [14]. Similar to VRD, VROID represents the detected visual relationships with relationship triplets and localize the subject and object for each relationship triplet by bounding boxes; nevertheless, it only focuses on the essential visual relationships which can describe the main content of an image, named "visual relation of interest" (VROI). Compared to IOID, VROID is more challenging because it requires to measure essentiality of visual relationships rather than that of instances as in IOID. To tackle the technical challenges of VROID, we proposed an Interest Propagation Network (IPNet), which contains a Panoptic Object Detection (POD) module, a Pair Interest Prediction module, and a Predicate Interest Prediction module. In the IPNet, interest is propagated from instances to pairs in the Pair Interest Prediction module and further propagated from pairs to triplets by combining outputs of pair interest prediction and predicate interest prediction.

The artifacts include the ViROI dataset and the source code of the IPNet model with variances. The source code is available at https://github.com/njumagus/VROID, and the related download links are provided in the README file.

1.2 Dataset

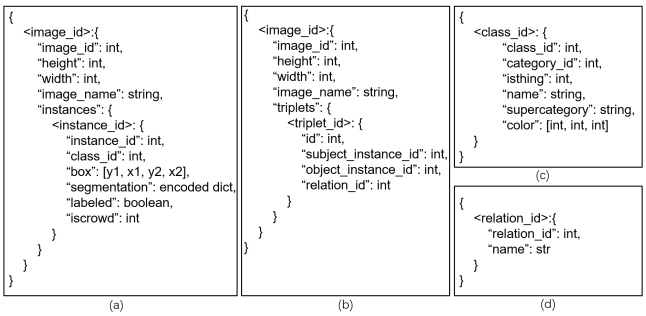

Figure 1: Data format in the json files. (a) The format of the "train_images_dict.json" and "test_images_dict.json" file. (b) The format of the "train_images_triplets_dict.json" and "test_images_triplets_dict.json" file. (c) The format of the "class_dict.json" file. (d) The format of the "relation_dict.json" file.

We constructed a ViROI dataset for VROID on the basis of 45,000 images in the IOID dataset [14] and their corresponding captions in the MSCOCO dataset [4]. After the images without VROIs are filtered out, the ViROI dataset contains 30,120 images, and is further divided into the training set (25,091 images with 91,496 VROIs) and the test set (5,029 images with 18,268 VROIs). The link for downloading the ViROI dataset is available in the README file.

The data of images and annotations are stored in six json files: "train_images_dict.json", "test_images_dict.json", "train_images_triplets_dict.json", "test_images_triplets_dict.json", "class_dict.json", and "relation_dict.json". The data format is shown in the Figure 1.

*Corresponding author.

Specifically, the <image_id> and <instance_id> in Figure 1(a) are both string values when working as the the keys in json files, representing the id of an image and an instance, respectively; the <image_id> and <triplet_id> in Figure 1(b) are both string values, representing the id of an image and a triplet, respectively; the <class_id> in Figure 1(c) is a string value, representing the "class_id" of an object category; the <relation_id> in Figure 1(d) is a string value, representing the "relation_id" of a predicate category. Same as that in the IOID dataset, the "category_id" is not contiguous in the category information provided by MSCOCO, and we give a contiguous "class_id" to each category. Figure 2 shows a visualization example of annotation result in the ViROI dataset. The VROIs in format of <subject, predicate, object> are listed under the image, and subjects and objects of the VROIs are labeled with bounding boxes.

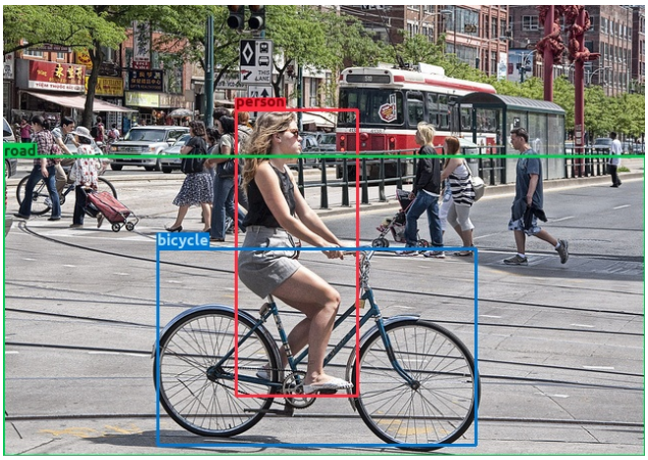

Figure 2: A visualization example of annotation result in the ViROI dataset.

1.3 Source code structure

The file structure of the source code is shown in the Figure 3, and the implementation is based on the Detectron2 [10]. The descriptions of some important folders and files are listed as follows:

- **component_analysis_test.sh**: executing testing for component analysis.
- **evaluate.py**: performing evaluation with prediction results.
- **frequency.sh**: executing testing of frequency baseline.
- **init_predicate_matrix.py**: generating word embedding matrix.
- **main.py**: performing training and testing on our model and the variances.
- **configs**: containing configuration files.
- **detectron2**: containing main files for the framework.
  - **config**: containing the python files for configuration parsing and default values.
  - **data**: containing files for loading data.

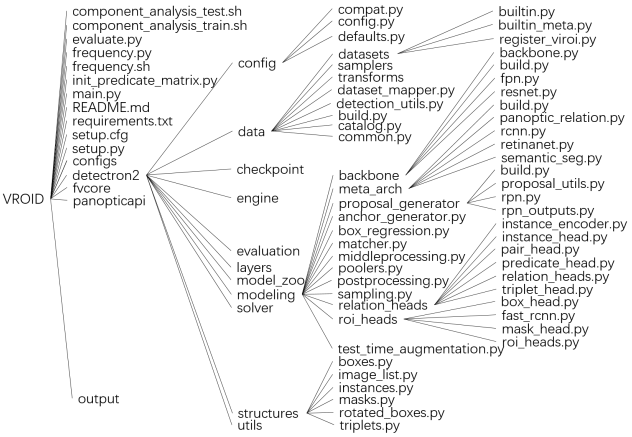

Figure 3: Structure of main files.

- **checkpoint**: containing files for saving checkpoints.
- **engine**: containing files for running the framework.
- **evaluation**: containing files for evaluation.
- **layers**: containing files for functions such as nms, roi_align and batch_norm.
- **model_zoo**: containing configuration files for different models.
- **modeling**: containing main files for constructing models.
  - **backbone**: containing files for network backbone.
  - **meta_arch**: containing main model files.
  - **proposal_generator**: containing files for proposal generation.
  - **relation_heads**: containing files for modules in IPNet.
    - **instance_encoder.py**: working for instance encoding.
    - **instance_head.py**: working for the instance interest prediction.
    - **pair_head.py**: working for pair interest prediction.
    - **predicate_head.py**: working for predicate interest prediction.
    - **relation_heads.py**: class for combining different interest prediction modules.
    - **triplet_head.py**: working as the triplet interest prediction module of the variance in the component analysis experiment.
  - **roi_heads**: containing files for the panoptic object detection module.
    - **box_head.py**: class for predicting object bounding boxes.
    - **fast_rcnn.py**: containing structures for the output of object detection.
    - **mask_head.py**: class for predicting object masks.
    - **roi_heads.py**: class for combining box and mask prediction.
  - **anchor_generator.py**: class for generating anchors.
  - **box_regression.py**: class for box transformation.
  - **matcher.py**: class for assigning each predicted element a ground-truth element.

    - **middleprocessing.py**: working for processing intermediate results.
    - **poolers.py**: working for ROI pooling.
    - **postprocessing.py**: working for processing final results.
    - **sampling.py**: working for sampling labels.
    - **test_time_augmentation.py**: working for test-time augmentation.
  - **solver**: containing files about the optimizer and the learning rate scheduler.
  - **structures**: containing files that define data structure.
  - **utils**: utilities in the framework.
- **fvcore**: containing the files in the fvcore, the dependency of the framework.
- **panopticapi**: containing the utilities for panoptic segmentation.
- **output**: containing output files, including log files, predicted files and saved weight files.

The main class defined in the "panoptic_relation.py" file combines the POD module with the relation modules. The backbone is defined in the "backbone" folder. The thing predictor and stuff predictor in the POD module are defined in the "semantic_seg.py" file and the "roi_heads" folder. The relation modules are defined in the "relation_heads" folder. The object generated from the POD module are encoded by the "instance_encoder.py" file and the object features are then input into the "instance_head.py" file. The features of object pairs and relation triplets are generated by object features and input into the "pair_head.py" and the "predicate_head.py" file.

Data loader is built in the "detectron2/data/build.py" file and the dataset is registered by the "builtin.py", "register_viroi.py" and "viroi.py" files. The data are transformed, mapped and rearranged by the "transforms" folder, the "dataset_mapper.py" and "detection_utils.py" files.

The "main.py" file contains training and test functions of the IPNet, as well as panoptic segmentation prediction. We also integrate the interest score computation in some variances: "triplet as output", "output with triplet", "output without pair", and "no instance".

2 EXPERIMENTS

2.1 Environment Installation

Compared to the original experimental settings in [14], we test the reproduced code in the following environment with higher performance hareware and newer version software:

(1) Operating system Ubuntu 18.04 LTS with CPU E5-2680 v4, GPU 3090, 64GB memory and 1TB free space.

(2) CUDA 11.1 and cuDNN 8.0.

(3) Python 3.8.3 with cython==0.29.21, matplotlib==3.2.2, Shapely ==1.7.1, termcolor==1.1.0, yacs==0.1.8, opencv_python==3.4.8.29, cloudpickle==1.5.0, numpy==1.20.0, pycocotools==2.0.2, iopath==0.1.7, tabulate==0.8.7, scipy==1.5.0, Pillow==8.1.0, setuptools==41.0.0, tensorboard==2.4.1, pyyaml==5.4.1.

We provide a docker image for all software dependencies, which is available for download in the README file.

2.2 Visualization demo

We provide a demo to visualize the result of our IPNet method when an image is given. The demo script can be performed as follows:

```
python main.py --config <configuration file path> --mode demo --image_path <image path> --visible --visible_num <N>
```

The "N" in the script represents the top-N results.

2.3 Custom training and inference

The model parameters defined in the "detectron2/config/defaults.py" file can be adjusted by a custom configuration file, and some important parameters as well as their description are shown in Table 1.

To train the IPNet model, we use the pretrained panoptic segmentation model with the ResNet101 as backbone. The training script can be performed as follows:

```
python main.py --config <configuration file path> --mode train_relation
```

The testing script can be performed as follows:

```
python main.py --config <configuration file path> --mode test_relation
```

To evaluate the results, the "evaluate.py" file can be performed with the following script:

```
python evaluate.py --pred_json <output file path> --top_n <K>
```

The "K" in the script is the same as the "K" in the metric *Recall@K*. Especially, when "K" is 0, the script outputs *Recall@*θ, where θ is the number of correct relations in the image.

2.4 Experiments in the original paper

For component analysis, we design 9 variances, and the experiments can be performed as follows:

```
./component_analysis_train.sh
./component_analysis_test.sh
```

The variances "output with triplet" and "triplet as output" share the same configuration file "triplet_as_output.yaml" and the outputs of the two variances are saved in "test_triplet_as_output.yaml.json" and "test_triplet_as_out-put.yaml_nopair.json" respectively. The variances "output without pair" and "output with instance" share the configuration file with "our", which is "our.yaml", and the results are saved in "test_our.yaml_no-pair.json" and "test_our.yaml_instance.json" respectively. The configuration file of variances "only raw predicate", "no instance", "no semantics features", "no locations features", and "bce loss" are "only_raw_predicate.yaml", "no_instance.yaml", "no_semantics_fea-tures.yaml", "no_locations_features.yaml" and "bce_loss.yaml" respectively and the results are saved in "test_only_raw_predicate.yaml.json", "test_no_instance.yaml.json", "test_no_semantics_features.yaml.json", "test_no_locations_features.yaml.json", and "test_bce_loss.yaml.json" respectively.

For comparison with other methods, the detection results for the baselines without detector need to be generated with followed script:

Table 1: Important parameters that can be customized.

Parameter	Description	Default Value
OUTPUT_DIR	The path of the output files from content root.	"./output"
DATASETS.TRAIN	The dataset for training.	"viroi_train"
DATASETS.TEST	The dataset for testing.	"viroi_test"
INPUT.MASK_FORMAT	The format of instance mask.	"bitmask"
SOLVER.IMS_PER_BATCH	The images of each batch.	1
SOLVER.CHECKPOINT_PERIOD	The iteration number for saving a checkpoint.	1000
SOLVER.MAX_ITER	The iteration number when stopping training.	120000
SOLVER.BASE_LR	The basic learning rate.	0.01
SOLVER.STEPS	The iteration numbers for decreasing learning rate.	(60000, 90000, 120000)
TEST.DETECTIONS_PER_IMAGE	The number of detected instance of each image.	80
MODEL.TRAINABLE	The trainable modules.	[]
MODEL.DEVICE	The device used for training.	"cuda"
MODEL.META_ARCHITECTURE	The python class for meta architecture.	"PanopticRelation"
MODEL.WEIGHTS	The pretrained model weights.	
MODEL.RESNETS.DEPTH	The depth of backbone resnet.	101
MODEL.RELATION_HEADS.INSTANCE_NUM	The number of instance categories including backbone.	134
MODEL.RELATION_HEADS.RELATION_NUM	The number of relation categories including no-relation.	250
MODEL.RELATION_HEADS.RELATION_HEAD_LIST	The modules used in the IPNet.	["instance", "pair", "predicate"]
MODEL.RELATION_HEADS.RELATION_INSTANCE_ENCODER.NAME	The class for instance encoding.	"InstanceEncoder1"
MODEL.RELATION_HEADS.RELATION_INSTANCE_HEAD.NAME	The class for instance of interest prediction.	"InstanceHead14"
MODEL.RELATION_HEADS.RELATION_PAIR_HEAD.NAME	The class for pair of interest prediction.	"PairHead17"
MODEL.RELATION_HEADS.RELATION_PREDICATE_HEAD.NAME	The class for predicate of interest prediction.	"PredicateHeadsMFULN45"

```
python main.py --config configs/VROID/
Base-Panoptic-FPN.yaml --mode test_panoptic
```

We provide the project links of related models of the baselines in the README file. We use 2 VRD baselines: STA [13] and MFURLN [17], 4 Scene Graph Generation (SGG) baselines: IMP [11], Graph R-CNN [12], neural motifs [16], and VCTree [9], and 2 baselines using VCTree, which works best with different salient object detection as postprocess among the above models: DSS [3] and NLDF [6]. Also, we design 2 baselines using ARNet [1] and MMT [2] for image captioning with Stanford CoreNLP Dependency Parser [7] for dependency parsing and DSG [8] for referring relationships. Especially, we provide a script for the baseline using frequency:

```
./frequency.sh
```

3 REPRODUCIBILITY EFFORTS

In the original paper, the description of some fields in the provided dataset is not clear and the software dependency list is not in accord with the description in README of the code repository. When the reviewers tried to run the scripts for the experiments, they found that the required environment is too new for most environments in use and errors were reported when the "no_instance" experiment was conducted. The reviewers also suggested that the authors should provide a visualization application.

In revision, the authors provided more details in the description of dataset files and conformed the software dependendies described in the paper and the README in code. The authors also quickly fixed the problems in the "no_instance" experiment. To make it convient for more researchers to run the code, the authors added a description of using custom environment in the README and provided a script to run the visualization demo. The reviewers acknowledged the efforts of the original authors to provide necessary corrections during the reviewing process, including adding detailed description of dataset, fixing minor bugs in the code, as well as providing convience for other researchers.

In conclusion, two reviewers and the authors worked together for this companion paper. The revised code now enables third-party researchers to reproduce the experiments in the original paper and is customizable for further research on visual relation detection.

4 CONCLUSION

In this paper, we provided the details of the artifacts of the paper "Visual Relation of Interest Detection" for replication. The artifacts contain the ViROI dataset and the source code for experiments in the paper.

ACKNOWLEDGEMENT

This work is supported by National Science Foundation of China (62072232, 61732007), Natural Science Foundation of Jiangsu Province (BK20191248), Science, Technology and Innovation Commission of Shenzhen Municipality (JCYJ20180307151516166), and Collaborative Innovation Center of Novel Software Technology and Industrialization.

REFERENCES

[1] Xinpeng Chen, Lin Ma, Wenhao Jiang, Jian Yao, and Wei Liu. 2018. Regularizing rnns for caption generation by reconstructing the past with the present. In *IEEE Conference on Computer Vision and Pattern Recognition*. 7995–8003.

[2] Marcella Cornia, Matteo Stefanini, Lorenzo Baraldi, and Rita Cucchiara. 2019. M^2: Meshed-Memory Transformer for Image Captioning. *arXiv preprint arXiv:1912.08226* (2019).

[3] Qibin Hou, Ming-Ming Cheng, Xiaowei Hu, Ali Borji, Zhuowen Tu, and Philip HS Torr. 2017. Deeply supervised salient object detection with short connections. In *IEEE Conference on Computer Vision and Pattern Recognition*. 3203–3212.

[4] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft coco: Common objects in context. In *European Conference on Computer Vision*. 740–755.

[5] Cewu Lu, Ranjay Krishna, Michael Bernstein, and Li Fei-Fei. 2016. Visual relationship detection with language priors. In *European Conference on Computer Vision*. 852–869.

[6] Zhiming Luo, Akshaya Mishra, Andrew Achkar, Justin A Eichel, Shaozi Li, and Pierremarc Jodoin. 2017. Non-local Deep Features for Salient Object Detection. In *IEEE Conference on Computer Vision and Pattern Recognition*.

[7] Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics System Demonstrations*. 55–60.

[8] Moshiko Raboh, Roei Herzig, Jonathan Berant, Gal Chechik, and Amir Globerson. 2020. Differentiable scene graphs. In *IEEE Winter Conference on Applications of Computer Vision*. 1488–1497.

[9] Kaihua Tang, Hanwang Zhang, Baoyuan Wu, Wenhan Luo, and Wei Liu. 2019. Learning to compose dynamic tree structures for visual contexts. In *IEEE Conference on Computer Vision and Pattern Recognition*. 6619–6628.

[10] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. 2019. Detectron2. https://github.com/facebookresearch/detectron2.

[11] Danfei Xu, Yuke Zhu, Christopher B Choy, and Li Fei-Fei. 2017. Scene graph generation by iterative message passing. In *IEEE Conference on Computer Vision and Pattern Recognition*. 5410–5419.

[12] Jianwei Yang, Jiasen Lu, Stefan Lee, Dhruv Batra, and Devi Parikh. 2018. Graph R-CNN for scene graph generation. In *European Conference on Computer Vision*. 670–685.

[13] Xu Yang, Hanwang Zhang, and Jianfei Cai. 2018. Shuffle-then-assemble: Learning object-agnostic visual relationship features. In *European Conference on Computer Vision*. 36–52.

[14] Fan Yu, Haonan Wang, Tongwei Ren, Jinhui Tang, and Gangshan Wu. 2019. Instance of Interest Detection. In *ACM International Conference on Multimedia*.

[15] Fan Yu, Haonan Wang, Tongwei Ren, Jinhui Tang, and Gangshan Wu. 2020. Visual Relation of Interest Detection. In *ACM International Conference on Multimedia*.

[16] Rowan Zellers, Mark Yatskar, Sam Thomson, and Yejin Choi. 2018. Neural motifs: Scene graph parsing with global context. In *IEEE Conference on Computer Vision and Pattern Recognition*. 5831–5840.

[17] Yibing Zhan, Jun Yu, Ting Yu, and Dacheng Tao. 2019. On Exploring Undetermined Relationships for Visual Relationship Detection. In *IEEE Conference on Computer Vision and Pattern Recognition*. 5128–5137.