Context R-CNN

Context R-CNN is an object detection model that uses contextual features to improve object detection. See https://arxiv.org/abs/1912.03538 for more details.

Table of Contents

Preparing Context Data for Context R-CNN

In this section, we will walk through the process of generating TfRecords with contextual features. We focus on building context from object-centric features generated with a pre-trained Faster R-CNN model, but you can adapt the provided code to use alternative feature extractors.

Each of these data processing scripts uses Apache Beam, which can be installed using

```
pip install apache-beam
```

and can be run locally, or on a cluster for efficient processing of large amounts of data. Note that generate_detection_data.py and generate_embedding_data.py both involve running inference, and may be very slow to run locally. See the Apache Beam documentation for more information, and Google Cloud Documentation for a tutorial on running Beam jobs on DataFlow.

Generating TfRecords from a set of images and a COCO-CameraTraps style JSON

If your data is already stored in TfRecords, you can skip this first step.

We assume a COCO-CameraTraps json format, as described on LILA.science.

COCO-CameraTraps is a format that adds static-camera-specific fields, such as a location ID and datetime, to the well-established COCO format. To generate appropriate context later on, be sure you have specified each contextual group with a different location ID, which in the static camera case would be the ID of the camera, as well as the datetime each photo was taken. We assume that empty images will be labeled 'empty' with class id 0.

To generate TfRecords from your database and local image folder, run

```
python
object_detection/dataset_tools/context_rcnn/create_cococameratraps_tfexample_main.py \
  --alsologtostderr \
  --output_tfrecord_prefix="/path/to/output/tfrecord/location/prefix" \
  --image_directory="/path/to/image/folder/" \
  --input_annotations_file="path/to/annotations.json"
```

Generating weakly-supervised bounding box labels for image-labeled data

If all your data already has bounding box labels you can skip this step.

Many camera trap datasets do not have bounding box labels, or only have bounding box labels for some of the data. We have provided code to add bounding boxes from a pretrained model (such as the Microsoft AI for Earth MegaDetector) and match the boxes to the image-level class label.

To export your pretrained detection model, run

```
python object_detection/export_inference_graph.py \
  --alsologtostderr \
  --input_type tf_example \
  --pipeline_config_path path/to/faster_rcnn_model.config \
  --trained_checkpoint_prefix path/to/model.ckpt \
  --output_directory path/to/exported_model_directory
```

To add bounding boxes to your dataset using the above model, run

```
python object_detection/dataset_tools/context_rcnn/generate_detection_data.py \
  --alsologtostderr \
  --input_tfrecord path/to/input_tfrecord@X \
  --output_tfrecord path/to/output_tfrecord@X \
  --model_dir path/to/exported_model_directory/saved_model
```

If an image already has bounding box labels, those labels are left unchanged. If an image is labeled 'empty' (class ID 0), we will not generate boxes for that image.

Generating and saving contextual features for each image

We next extract and store features for each image from a pretrained model. This model can be the same model as above, or be a class-specific detection model trained on data from your classes of interest.

To export your pretrained detection model, run

```
python object_detection/export_inference_graph.py \
  --alsologtostderr \
  --input_type tf_example \
  --pipeline_config_path path/to/pipeline.config \
  --trained_checkpoint_prefix path/to/model.ckpt \
  --output_directory path/to/exported_model_directory \
  --additional_output_tensor_names detection_features
```

Make sure that you have set `output_final_box_features: true` within your config file before exporting. This is needed to export the features as an output, but it does not need to be set during training.

To generate and save contextual features for your data, run

```
python object_detection/dataset_tools/context_rcnn/generate_embedding_data.py \
  --alsologtostderr \
  --embedding_input_tfrecord path/to/input_tfrecords* \
  --embedding_output_tfrecord path/to/output_tfrecords \
  --embedding_model_dir path/to/exported_model_directory/saved_model
```

Building up contextual memory banks and storing them for each context group

To build the context features you just added for each image into memory banks, run

```
python object_detection/dataset_tools/context_rcnn/add_context_to_examples.py \
  --input_tfrecord path/to/input_tfrecords* \
  --output_tfrecord path/to/output_tfrecords \
  --sequence_key image/location \
  --time_horizon month
```

where the input_tfrecords for add_context_to_examples.py are the output_tfrecords from generate_embedding_data.py.

For all options, see add_context_to_examples.py. By default, this code builds TfSequenceExamples, which are more data efficient (this allows you to store the context features once for each context group, as opposed to once per image). If you would like to export TfExamples instead, set flag `--output_type tf_example`.

If you use TfSequenceExamples, you must be sure to set `input_type: TF_SEQUENCE_EXAMPLE` within your Context R-CNN configs for both train_input_reader and test_input_reader. See `object_detection/test_data/context_rcnn_camera_trap.config` for an example.

Training a Context R-CNN Model

To train a Context R-CNN model, you must first set up your config file. See `test_data/context_rcnn_camera_trap.config` for an example. The important difference between this config and a Faster R-CNN config is the inclusion of a `context_config` within the model, which defines the necessary Context R-CNN parameters.

```
context_config {
      max_num_context_features: 2000
      context_feature_length: 2057
    }
```

Once your config file has been updated with your local paths, you can follow along with documentation for running locally, or on the cloud.

Exporting a Context R-CNN Model

Since Context R-CNN takes context features as well as images as input, we have to explicitly define the other inputs ("side_inputs") to the model when exporting, as below. This example is shown with default context feature shapes.

```
python export_inference_graph.py \
    --input_type image_tensor \
    --input_shape 1,-1,-1,3 \
    --pipeline_config_path /path/to/context_rcnn_model/pipeline.config \
    --trained_checkpoint_prefix /path/to/context_rcnn_model/model.ckpt \
    --output_directory /path/to/output_directory \
    --use_side_inputs True \
    --side_input_shapes 1,2000,2057/1 \
    --side_input_names context_features,valid_context_size \
    --side_input_types float,int
```

If you have questions about Context R-CNN, please contact Sara Beery.