Image categories MobileNetv2

1.Function Introduction

The mobilenetv2 image classification algorithm example uses images as input and utilizes the BPU for inference. It publishes messages containing object categories.

The mobilenetv2 is a caffe model trained on the ImageNet data dataset. The model source can be found at: https://github.com/shicai/MobileNet-Caffe . Supported target types include people, animals, fruits, vehicles, and other 1000 categories. For specific supported categories, please refer to the file /opt/tros/lib/dnn_node_example/config/imagenet.list on the RDK (TogetheROS.Bot installed).

Code repository： https://github.com/D-Robotics/hobot_dnn

Applications: mobilenetv2 is capable of predicting the category of a given image, and can be used for tasks such as digit recognition and object recognition. It is mainly applied in fields such as text recognition and image retrieval.

Food type identification case: https://github.com/frotms/Chinese-and-Western-Food-Classification

2.Preparation

- RDK has been flashed with the provided Ubuntu 20.04/22.04 system image.
- The tros.b has been successfully installed on RDK.
- MIPI or USB camera has been installed on RDK. If there is no camera available, algorithm effects can be experienced by local JPEG/PNG images or MP4, H.264, and H.265 videos offline.
- Ensure the PC can access RDK through the network.

3.User Guide

3.1Using a USB camera

Subscribe to the images published by the sensor package for mobilenetv2 image classification. After inference, the algorithm message will be published, and the image with corresponding algorithm result will be displayed on the PC browser through the websocket package.

- Enable USB camera node

```
# Configure the tros.b environment
source /opt/tros/humble/setup.bash
```

```
# Configuring USB camera
export CAM_TYPE=usb

# Start the launch file
ros2 launch dnn_node_example dnn_node_example.launch.py
dnn_example_config_file:=config/mobilenetv2workconfig.json
dnn_example_image_width:=480 dnn_example_image_height:=272
```

- Output Results

The terminal output during execution shows the following information:

```
[example-3] [WARN] [1655095481.707875587] [example]: Create ai msg publisher with
topic_name: hobot_dnn_detection
[example-3] [WARN] [1655095481.707983957] [example]: Create img
hbmem_subscription with topic_name: /hbmem_img
[example-3] [WARN] [1655095482.985732162] [img_sub]: Sub img fps 31.07
[example-3] [WARN] [1655095482.992031931] [example]: Smart fps 31.31
[example-3] [WARN] [1655095484.018818843] [img_sub]: Sub img fps 30.04
[example-3] [WARN] [1655095484.025123362] [example]: Smart fps 30.04
[example-3] [WARN] [1655095485.051988567] [img_sub]: Sub img fps 30.01
[example-3] [WARN] [1655095486.057854228] [example]: Smart fps 30.07
```

The log shows that the topic for publishing the algorithm inference results is `hobot_dnn_detection`, and the topic for subscribing to images is `/hbmem_img`.

On the PC side, enter http://IP:8000 in the browser to view the image and the rendering effect of the algorithm (where IP is the IP address of the RDK):

3.2Using IMX219 camera

Subscribe to the images published by the sensor package for mobilenetv2 image classification. After inference, the algorithm message will be published, and the image with corresponding algorithm result will be displayed on the PC browser through the websocket package.

- Enable MIPI camera node

```
# Configure the tros.b environment
source /opt/tros/humble/setup.bash
```

```
# Configuring MIPI camera
export CAM_TYPE=mipi
# Start the launch file
ros2 launch dnn_node_example dnn_node_example.launch.py
dnn_example_config_file:=config/mobilenetv2workconfig.json
dnn_example_image_width:=480 dnn_example_image_height:=272
```

- Output Results

The terminal output during execution shows the following information:

```
[example-3] [WARN] [1655095481.707875587] [example]: Create ai msg publisher with
topic_name: hobot_dnn_detection
[example-3] [WARN] [1655095481.707983957] [example]: Create img
hbmem_subscription with topic_name: /hbmem_img
[example-3] [WARN] [1655095482.985732162] [img_sub]: Sub img fps 31.07
[example-3] [WARN] [1655095482.992031931] [example]: Smart fps 31.31
[example-3] [WARN] [1655095484.018818843] [img_sub]: Sub img fps 30.04
[example-3] [WARN] [1655095484.025123362] [example]: Smart fps 30.04
[example-3] [WARN] [1655095485.051988567] [img_sub]: Sub img fps 30.01
[example-3] [WARN] [1655095486.057854228] [example]: Smart fps 30.07
```

The log shows that the topic for publishing the algorithm inference results is `hobot_dnn_detection`, and the topic for subscribing to images is `/hbmem_img`.

On the PC side, enter http://IP:8000 in the browser to view the image and the rendering effect of the algorithm (where IP is the IP address of the RDK):

3.1.2Local image reflow

The mobilenetv2 image classification algorithm example can use local JPEG/PNG format images offline. After inference, the algorithm renders the resulting image and saves it in the local path.

- Running the recharge node

```
# Configure the tros.b environment
source /opt/tros/humble/setup.bash
```

```
# Start the launch file
ros2 launch dnn_node_example dnn_node_example_feedback.launch.py
dnn_example_config_file:=config/mobilenetv2workconfig.json
dnn_example_image:=config/target_class.jpg
```

- Recharge results

The following information is outputted in the terminal:

```
[example-1] [INFO] [1654767648.897132079] [example]: The model input width is 224
and height is 224
[example-1] [INFO] [1654767648.897180241] [example]: Dnn node feed with local
image: config/target_class.jpg
[example-1] [INFO] [1654767648.935638968] [example]: task_num: 2
[example-1] [INFO] [1654767648.946566665] [example]: Output from image_name:
config/target_class.jpg, frame_id: feedback, stamp: 0.0
[example-1] [INFO] [1654767648.946671029] [ClassificationPostProcess]: outputs
size: 1
[example-1] [INFO] [1654767648.946718774] [ClassificationPostProcess]: out cls
size: 1
[example-1] [INFO] [1654767648.946773602] [ClassificationPostProcess]: class
type:window-shade, score:0.776356
[example-1] [INFO] [1654767648.947251721] [ImageUtils]: target size: 1
[example-1] [INFO] [1654767648.947342212] [ImageUtils]: target type: window-
shade, rois.size: 1
[example-1] [INFO] [1654767648.947381666] [ImageUtils]: roi.type: , x_offset: 112
y_offset: 112 width: 0 height: 0
[example-1] [WARN] [1654767648.947563731] [ImageUtils]: Draw result to file:
render_feedback_0_0.jpeg
```

The log shows that the algorithm infers that the image `config/target_class.jpg` is classified as a window-shade with a confidence score of 0.776356 (the algorithm only outputs the highest confidence classification result). The rendered image is stored with the file name `render_feedback_0_0.jpeg`, and the rendered image looks like this: