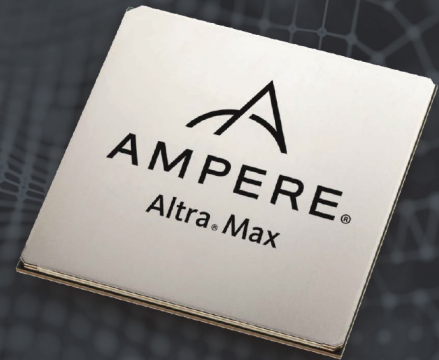




# Deploying AIO on Tencent Cloud for AI Inference Usages



Tencent CVM standard SR1 is the first new generation of CVM standard computing instance specifications launched by Tencent Cloud equipped with ARM architecture processors. SR1 is based on the Ampere® Altra® processor with an all-core consistent operating frequency of 3.0 GHz. The number of instance cores ranges from 1 core to 64 cores and supports 1:2, 1:4, and other processors/memory ratios.

Compared with the x86 architecture instance, the users get excellent value for money. Ampere provides an optimized AI framework (Ampere AI) for the SR1 instance based on the Ampere Altra processor and provides free images for customers to use through the Tencent Image Marketplace.

This article will introduce how to create an SR1 instance on Tencent Cloud and evaluate the performance of computer vision classification based on TensorFlow.

## AI Inference with CPU

Presently, there are three main sources of computing power for AI reasoning applications, namely CPU+AI dedicated chips, CPU+GPU, and pure CPU. According to the AI hardware insights report previously released by Statista and McKinsey, CPU-based inference currently accounts for more than 50%. Compared to the other two modes, there are several main reasons for using CPU for AI inference:

- More portable and no vendor lock
- Fewer complexities involving operating systems, drivers, runtime libraries, and more
- Continuous innovations in AI model algorithms (e.g., sparsity, quantization, etc.) on the CPU can provide high throughput close to the GPU.
- Easier to scale out and integrate with other software stacks.

More importantly, building inference applications on the CPU can easily integrate AI into business logic modules and integrate into the Cloud Native system of microservices.

## Create SR1 Instance

Create a 16vCPU SR1 instance SR1.4XLARGE32 for evaluation, which is configured with 16 Ampere Altra physical cores and 32GB of memory.

1. Log into Tencent Cloud's console.
2. Select **New** under the Instance category, and you will enter the instance creation page.
3. Since SR1 is currently only available in Guangzhou Sixth District, select **Guangzhou > Guangzhou Sixth District > Standard SR1**.

快速配置 自定义配置

1. 选择机型 2. 设置主机 3. 确认配置信息

计费模式  包年包月  按量计费  竞价实例 [② 详细对比](#)

地域

华南地区 华东地区 华北地区 西南地区 港澳台地区

广州 上海 南京 **NEW** 北京 成都 重庆 中国香港

亚太东北 亚太东南 美国西部 欧洲地区

首尔 东京 新加坡 曼谷 雅加达 **NEW** 硅谷 法兰克福 莫斯科

亚太南部 美国东部 南美地区 北美地区

孟买 弗吉尼亚 圣保罗 **NEW** 多伦多 [② 更多地域](#)

不同地域云产品之间内网不互通；选择最靠近您客户的地域，可降低访问时延。创建成功后不支持切换地域。 [查看我的云服务器地域](#) [详细对比](#)

可用区  随机可用区  广州三区  广州四区  广州六区 **NEW**  广州七区 **NEW** [②](#)

网络 vpc-4d1bm70r | Default-VPC (默认) | 17: subnet-07fnur14 | Default-Subnet (默认) [②](#) [子网剩余可用IP 4080个](#)

当前网络为默认私有网络/子网，建议您根据业务需要进行调整  
如现有私有网络/子网不符合您的要求，可以去控制台 [新建私有网络](#) 或 [新建子网](#)。云服务器购买后可以通过控制台切换私有网络完成私有网络/子网的切换

实例

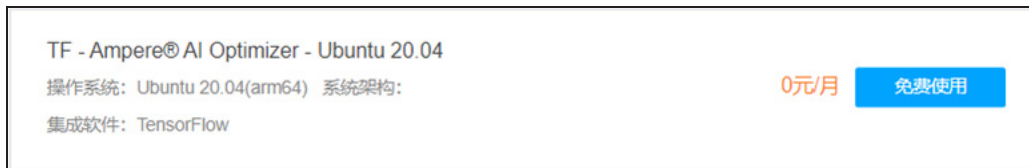
全部机型  标准型  高IO型  内存型  计算型  GPU机型  FPGA机型  大数据型  裸金属云服务器 [②](#)

全部实例类型  标准型S6  标准型SA2  标准型S5  标准存储增强型S5se **NEW**  标准型SA3 **NEW**  标准型SR1 **NEW**  标准型S4

4. Then you will see SR1 instances of different specifications, choose the instance of SR1.4XLARGE32 specification.

<input checked="" type="radio"/>	标准型SR1	SR1.4XLA...	16核	32GB	Ampere Altra(2.8 GHz)	6Gbps	110万PPS
<input type="radio"/>	标准型SR1	SR1.8XLA...	32核	64GB	Ampere Altra(2.8 GHz)	12Gbps	220万PPS
<input type="radio"/>	标准型SR1	SR1.16XL...	64核	128GB	Ampere Altra(2.8 GHz)	23Gbps	450万PPS

- For an OS image, select Image Marketplace > Select from Marketplace, then search for Ampere, and select TF - Ampere® AI Optimizer - Ubuntu 20.04 image to use Ampere's SR1-optimized TensorFlow 2.7 and free examples.



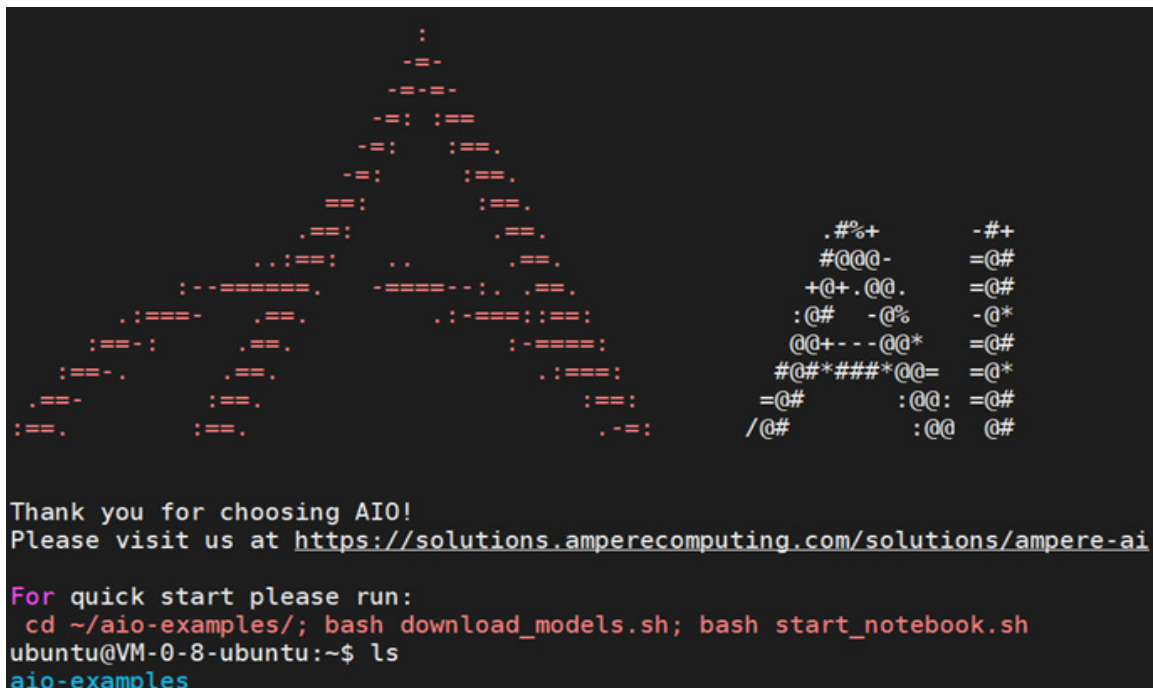
- After setting other instance configurations, you can confirm the configuration and activate the instance.

## Start and Connect to the Instance

After creating the instance, you can start and log in. You can obtain the IP address of the instance from the console. Depending on the login method set during creation, you can log in to the instance either using a password or a key.

```
$ ssh ubuntu@<public ip address of the instances>
```

After logging in, a welcome screen of Ampere AI is displayed.



This image not only integrates Ampere-optimized TensorFlow, but it also contains lots of example code, which can also be obtained from [Github](#).

## Running the TensorFlow AIO Example

TensorFlow is an end-to-end open-source machine learning platform. It has a comprehensive and flexible ecosystem of tools, libraries, and community resources that enable researchers to improve machine learning techniques and developers to easily build and deploy machine learning powered applications.

The OS image chosen from the image marketplace already includes TensorFlow 2.7 optimized for the Ampere Altra CPU. To run the sample program provided by aio-example, you need to download the model first. aio-examples provide different models, including image classification and object detection with 32-bit, 16-bit, and 8-bit models.

```
$ cd aio-examples
$ ./download_models.sh
```

Depending on the network conditions, downloading all models will take a few minutes. Use the TensorFlow ResNet\_50\_v15 classification model for testing and evaluation. ResNet-50 is one of the most used classification models for images.

Since the Ampere Altra CPU is single-core and single-threaded, each vCPU in SR1 corresponds to an Altra physical core, so when testing with SR1.4XLARGE32, we specify AIO\_NUM\_THREADS as 16. We first test the double-precision model for FP32.

```
cd classification/resnet_50_v15
export AIO_NUM_THREADS=16
python3 run.py -m resnet_50_v15_tf_fp32.pb -p fp32
```

```
ubuntu@VM-0-21-ubuntu:~/aio-examples/classification/resnet_50_v15$ python3 run.py -m resnet_50_v15_tf_fp32.pb -p fp32
/usr/lib/python3/dist-packages/requests/__init__.py:89: RequestsDependencyWarning: urllib3 (1.26.9) or chardet (3.0.4)
oesn't match a supported version!
  warnings.warn("urllib3 ({}), or chardet ({}), doesn't match a supported "

Running with 16 threads

Top-1 accuracy = 0.800
Top-5 accuracy = 1.000

Accuracy figures above calculated on the basis of 5 images.

Latency: 15 ms
Throughput: 65.36 ips
```

With 16 cores, ResNet\_50\_v15 can process 65.36 images per second (65.36 ips) with a latency of 15 ms.

Next, we test the FP16-based model.

```
cd classification/resnet_50_v15
export AIO_NUM_THREADS=16
python3 run.py -m resnet_50_v15_tf_fp16.pb -p fp16
```

```
ubuntu@VM-0-21-ubuntu:~/aio-examples/classification/resnet_50_v15$ python3 run.py -m resnet_50_v15_tf_fp16.pb -p fp16
/usr/lib/python3/dist-packages/requests/__init__.py:89: RequestsDependencyWarning: urllib3 (1.26.9) or chardet (3.0.4)
oesn't match a supported version!
  warnings.warn("urllib3 ({}), or chardet ({}), doesn't match a supported "

Running with 16 threads

Top-1 accuracy = 0.800
Top-5 accuracy = 1.000

Accuracy figures above calculated on the basis of 5 images.

Latency: 9 ms
Throughput: 115.59 ips
```

Models with FP16 offer up to 115.59 ips of throughput, and that is because the Ampere Altra processors have native support for FP16. Compared to the FP32 model, the FP16 model can provide nearly 2x image processing capacity without accuracy loss.

## Performance Comparison with Other Instances

The aio-example here can also run on Tencent CVM instances based on Intel CPUs and AMD CPUs. We created 16vCPU instances- S6.4XLARGE32 and SA3.4XLARGE32. The S6.4XLARGE32 is a 16vCPU instance based on Intel® Xeon® Ice Lake processors, and SA3.4XLARGE32 is a 16vCPU instance based on AMD EPYC™ Milan processors. Unlike SR1.4XLARGE32, the 16vCPU on x86 is 16 threads, not physical cores, and the actual physical core is 8.

We run [Intel-TensorFlow](#) on S6.4XLARGE32. Intel-TensorFlow is Intel-optimized TensorFlow to take full advantage of the AVX-512 instruction set.

AMD also provides [ZenDNN](#) optimized for AMD CPU, but the test results with ZenDNN in Tencent CVM are not better than native TensorFlow, so the following SA3.4XLARGE32 data uses native TensorFlow.

The performance of the "ResNet\_50\_v15" model on the three CVM instances is as below.

Table 1: Performance of ResNet\_50\_v15 Model on the CVM Instances

INSTANCE TYPE	VCPU NUMBER	PRICE(CNY/HOUR)	MODEL	IPS	IPS/PRICE	LATENCY(MS)
S6.4xLARGE32	16	2.51	fp32	47	18.73	21
SA3.4xLARGE32	16	1.88	fp32	44	23.29	23
SR1.4xLARGE32	16	2.04	fp32	65	32.04	15
SR1.4xLARGE32	16	2.04	fp16	116	56.66	9

We can see that the number of images processed per second (ips), SR1.4xLARGE32 is 40% and 50% higher than that of S6 and SA3 instances of the same shape; if we consider the price difference of a single instance, at the same price, SR1 .4xLARGE32 can achieve 70% and 40% higher performance than the S6 and SA3 of the same shape.

At the same time, the SR1 instance provides support for FP16, which can achieve higher throughput and lower latency.

## Visual Example for Jupyter Notebook

aio-example also provides Jupyter Notebook scripts for editing, debugging, and visualizing. We will walk through the object detection model SSD Inception v2 example:

1. Start Jupyter Notebook in CVM.

```
ubuntu@VM-0-21-ubuntu:~$ cd aio-examples/
ubuntu@VM-0-21-ubuntu:~/aio-examples$ ./start_notebook.sh
On your local system please open a new terminal window and run:
ssh -N -L 8080:localhost:8080 -i ./your_key.key your_user@xxx.xxx.xxx.xxx

After that open one of the links printed out below in your local browser

[I 14:13:26.969 NotebookApp] Writing notebook server cookie secret to /home/ubuntu/.local/share/jupyter/runtime/notebook_cookie_secret
[I 14:13:27.475 NotebookApp] Serving notebooks from local directory: /home/ubuntu/aio-examples
[I 14:13:27.475 NotebookApp] Jupyter Notebook 6.4.11 is running at:
[I 14:13:27.475 NotebookApp] http://localhost:8080/?token=1efe6b3850396a796da4804adf73b2e758a62b282941a6ee
[I 14:13:27.475 NotebookApp] or http://127.0.0.1:8080/?token=1efe6b3850396a796da4804adf73b2e758a62b282941a6ee
[I 14:13:27.476 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 14:13:27.479 NotebookApp]

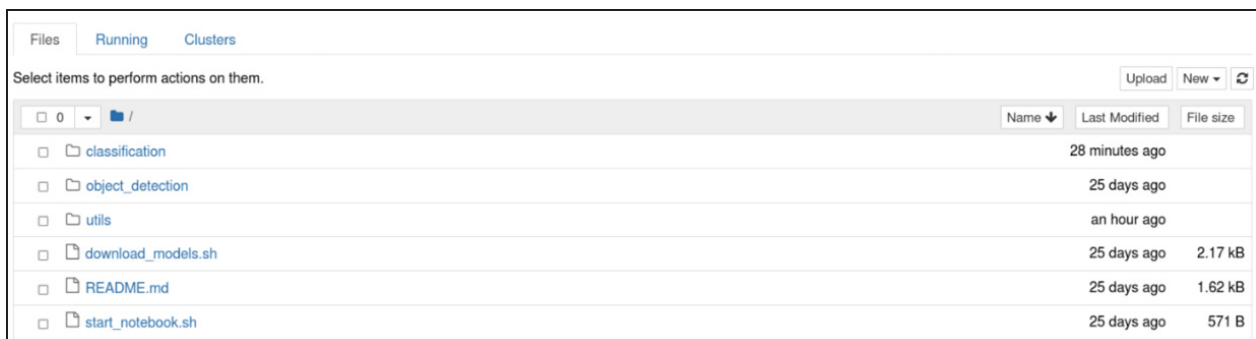
To access the notebook, open this file in a browser:
file:///home/ubuntu/.local/share/jupyter/runtime/nbserver-6171-open.html
Or copy and paste one of these URLs:
http://localhost:8080/?token=1efe6b3850396a796da4804adf73b2e758a62b282941a6ee
or http://127.0.0.1:8080/?token=1efe6b3850396a796da4804adf73b2e758a62b282941a6ee
```

2. On another machine with a browser, execute the following command.

```
ssh -N -L 8080:localhost:8080 ubuntu@<public ip address of the instances>
```

3. Enter the password of the instance and open the ssh tunnel.
4. Enter the following address in a browser to see AIO's Jupyter Notebook:

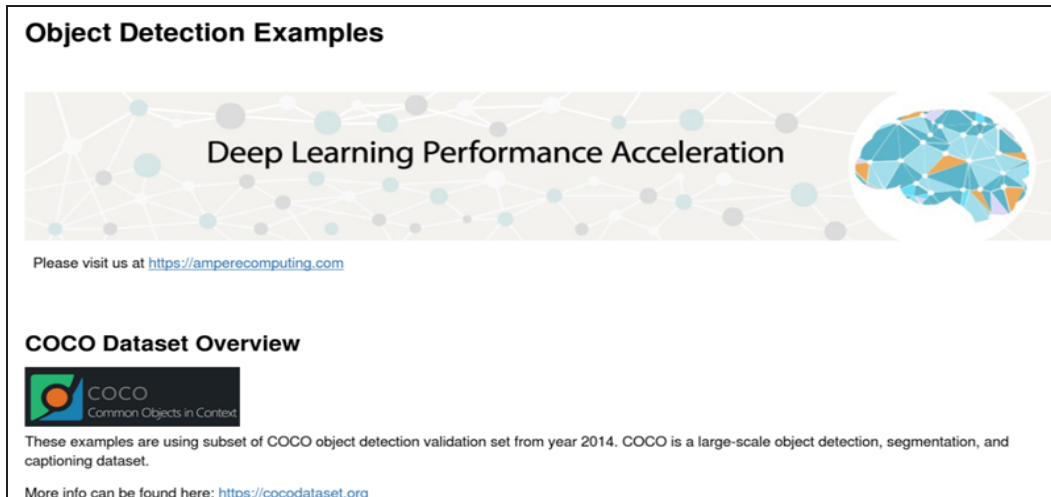
```
https://localhost:8080/?token=1efe6b3850396a796da4804adf73b2e758a62b282941a6ee
or
https://127.0.0.1:8080/?token=1efe6b3850396a796da4804adf73b2e758a62b282941a6ee
```



The screenshot shows the Jupyter Notebook interface with the 'Files' tab selected. The file browser displays a directory structure with folders and files. The folders are 'classification', 'object\_detection', and 'utils'. The files are 'download\_models.sh', 'README.md', and 'start\_notebook.sh'. The 'Last Modified' column shows the time since each item was last modified, and the 'File size' column shows the size of the files.

Name	Last Modified	File size
classification	28 minutes ago	
object_detection	25 days ago	
utils	an hour ago	
download_models.sh	25 days ago	2.17 kB
README.md	25 days ago	1.62 kB
start_notebook.sh	25 days ago	571 B

5. Select `object_detection > examples.ipynb`, to view the Object Detection Examples page.



**Object Detection Examples**

Deep Learning Performance Acceleration

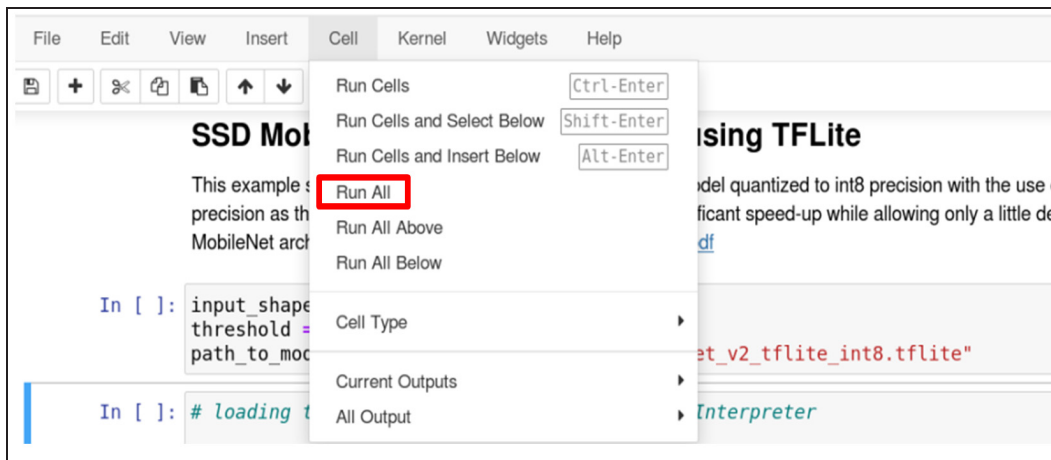
Please visit us at <https://amperecomputing.com>

**COCO Dataset Overview**

These examples are using subset of COCO object detection validation set from year 2014. COCO is a large-scale object detection, segmentation, and captioning dataset.

More info can be found here: <https://cocodataset.org>

6. Click `Cell > Run All` to run the code.



File Edit View Insert Cell Kernel Widgets Help

Run Cells `Ctrl-Enter`

Run Cells and Select Below `Shift-Enter`

Run Cells and Insert Below `Alt-Enter`

**Run All**

Run All Above

Run All Below

Cell Type

Current Outputs

All Output

**SSD Mo**

This example s  
precision as th  
MobileNet arch

```
In [ ]: input_shape  
        threshold =  
        path_to_mod
```

**using TFLite**

del quantized to int8 precision with the use of  
ficant speed-up while allowing only a little deg  
df

```
et_v2_tflite_int8.tflite"
```

Interpreter

7. Check the result.

```
In [40]: # running the model with AIO enabled in fp16 precision
tf.AIO.force_enable()

with tf.compat.v1.Session(config=config, graph=graph) as sess:
    # warm-up run
    _ = sess.run(output_dict, feed_dict)

    # actual run
    start = time.time()
    output_aio = sess.run(output_dict, feed_dict)
    finish = time.time()

latency_ms = (finish - start) * 1000
print("\nSSD Inception v2 FP16 latency with AIO: {:.0f} ms\n".format(latency_ms))

SSD Inception v2 FP16 latency with AIO: 23 ms
```

```
In [41]: # running the model with AIO disabled in fp16 precision
tf.AIO.force_disable()

with tf.compat.v1.Session(config=config, graph=graph) as sess:
    # warm-up run
    _ = sess.run(output_dict, feed_dict)

    # actual run
    start = time.time()
    output_no_aio_fp16 = sess.run(output_dict, feed_dict)
    finish = time.time()

latency_ms = (finish - start) * 1000
print("\nSSD Inception v2 FP16 latency without AIO: {:.0f} ms\n".format(latency_ms))

SSD Inception v2 FP16 latency without AIO: 2009 ms
```

```
In [43]: # running the model with AIO disabled in fp32 precision
tf.AIO.force_disable()

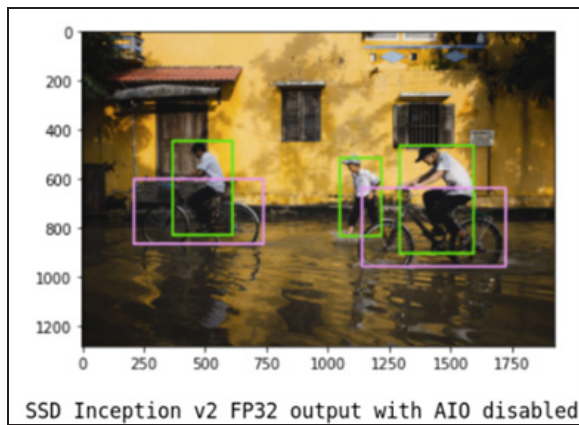
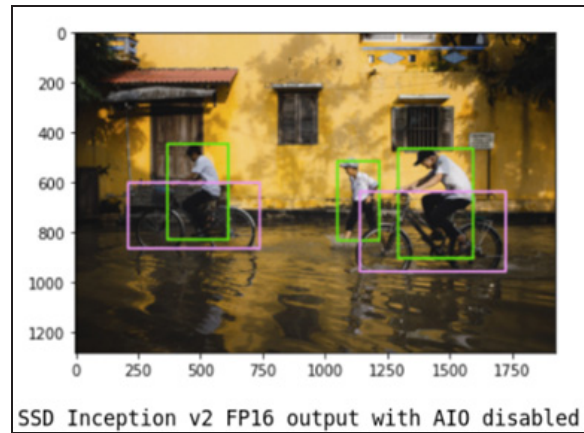
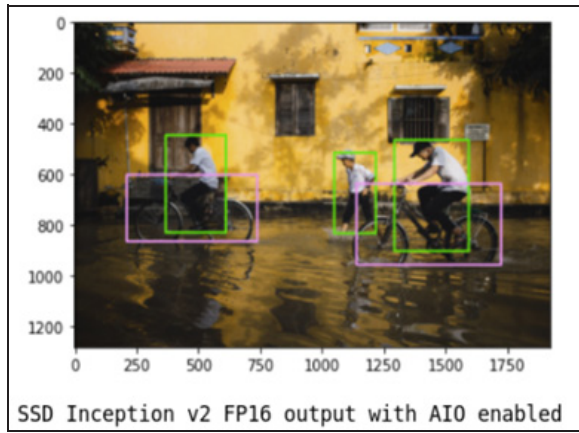
with tf.compat.v1.Session(config=config, graph=graph) as sess:
    # warm-up run
    _ = sess.run(output_dict, feed_dict)

    # actual run
    start = time.time()
    output_no_aio_fp32 = sess.run(output_dict, feed_dict)
    finish = time.time()

latency_ms = (finish - start) * 1000
print("\nSSD Inception v2 FP32 latency without AIO: {:.0f} ms\n".format(latency_ms))

SSD Inception v2 FP32 latency without AIO: 76 ms
```





Similarly, you can also run other examples in aio-examples.

## Summary

The Tencent SR1 instance is powered by Ampere Altra Cloud Native processor, featuring scalable performance, consistent operating frequencies, and single-threaded cores. Combining with Ampere optimized frameworks, SR1 can achieve a price-performance advantage of 1.7x over x86-based instances in AI inference.

In addition to the free images on the Tencent Cloud Marketplace, users can also obtain ready-to-use docker images, including code and documentation, from the [Ampere Solutions website](#), which can be downloaded after accepting the End User License Agreement. The docker image contains a standard ML framework (TensorFlow, PyTorch, ONNX, etc.) pre-installed with optimized software to run inference scripts on Tencent CVM SR1 without changes. Example models for image classification and object detection are also provided in the image.

Furthermore, Ampere offers Ampere Model Library (AML). It contains a number of ready-to-use Computer Vision, Natural Language Processing (NLP), and Recommender models. AML can be accessed on GitHub: [https://github.com/AmpereComputingAI/ampere\\_model\\_library](https://github.com/AmpereComputingAI/ampere_model_library).

The [Ampere Computing Trial Access Program](#) is open for SR1 instances application. Developers who are interested can try SR1 instances for free through this program.

## Resource Links

- <https://solutions.amperecomputing.com/solutions/ampere-ai>
- <https://github.com/AmpereComputingAI/aio-examples>
- <https://cloud.tencent.com/document/product/213/55669>
- <https://developer.amd.com/zendnn/>