rocAL Documentation

Release 2.4.0

Advanced Micro Devices, Inc.

Nov 26, 2025

INSTALLATION

1 rocAL prerequisites 3

2 Installing rocAL with the package installer 5
2.1 Basic installation . . . 5
2.2 Complete installation . . . 6

3 Building and installing rocAL from source code 7

4 rocAL Overview 9
4.1 Key Components . . . 10
4.2 Third-party Integration . . . 10
4.3 rocAL Operators . . . 10

5 rocAL Architecture components 15
5.1 rocAL Master-Graph . . . 15
5.2 ROCm Performance Primitive (RPP) Library . . . 16

6 Using rocAL with PyTorch for training 17

7 Using rocAL with TensorFlow for training 19

8 Using rocAL with JAX for training 21

9 rocAL examples 23

10 rocAL RNNT dataloading in Python 25

11 Using rocAL with C++ API 27
11.1 C++ Common APIs . . . 27

12 rocAL C++ API 31

13 rocAL Python API overview 33

14 rocAL Python API 35

15 License 37

The ROCm Augmentation Library (rocAL) lets you improve the throughput and performance of your deep learning applications. It's designed to efficiently decode and process image and video pipelines from a variety of storage formats on AMD GPUs and CPUs. Its C++ and Python APIs let you program customizable pipelines for different datasets and models. rocAL is optimized for loading and pre-processing data for deep learning applications, with support for multiple data formats and augmentations.

The rocAL public repository is located at https://github.com/ROCm/rocAL.

Installation

- *rocAL prerequisites*
- *Installing rocAL with the package installer*
- *Building and installing rocAL from source*

Conceptual

- *rocAL Overview*
- *rocAL Architecture components*

How to

- *Run PyTorch training with rocAL*
- *Run TensorFlow training with rocAL*
- *Run JAX training with rocAL*

Examples

- *rocAL framework integration examples*

Reference

- *rocAL RNNT dataloading*
- *rocAL C++ API overview*
- *rocAL C++ reference*
- *rocAL Python API overview*
- *rocAL Python reference*

To contribute to the documentation refer to Contributing to ROCm Docs.

You can find licensing information on the Licensing page.

CHAPTER

ONE

ROCAL PREREQUISITES

rocAL requires ROCm running on GPUs based on the CDNA architecture installed with the AMDGPU installer.

rocAL has been tested on the following Linux environments:

- Ubuntu 22.04 and 24.04
- RHEL 8 and 9
- SLES 15 SP7

See Supported operating systems for the complete list of ROCm supported Linux environments.

Building rocAL from source requires CMake Version 3.10 or later, AMD Clang++ Version 18.0.0 or later, and the following compiler support:

- C++17
- OpenMP
- Threads

Most prerequisites are installed with the *package installer*.

When building rocAL from source, the `rocAL-setup.py` setup script can be used to install prerequisites:

```
rocAL-setup.py [-h] [--directory DIRECTORY; default ~/] \
                    [--rocm_path ROCM_PATH; default /opt/rocm] \
                    [--backend HIP|OCL; default HIP] \
                    [--ffmpeg ON|OFF; default OFF] \
                    [--reinstall ON|OFF; default OFF]
```

The following prerequisites are required and are installed with both the package installer and the setup script:

- MIVisionX with AMD OpenVX and the VX_RPP and AMD Media extensions
- The half-precision floating-point library version 1.12.0 or later
- Google Protobuf version 3.12.4 or later
- LMBD Library
- TurboJPEG
- PyBind11 version 2.11.1
- RapidJSON
- OpenCV
- Python3, Python3 pip, and Python3 wheel

libstdc++-12-dev is required on Ubuntu 22.04 only and must be installed manually.

FFMPEG is not required, but is installed by the package installer. It can also be installed with the setup script using the `--ffmpeg` option.

rocDecode and rocJPEG are installed by the package installer and the setup script, but aren't required by rocAL. When installed, rocJPEG is used as the hardware image decoder and rocDecode is used as the hardware video decoder.

Note

TurboJPEG must be installed manually on SLES.
To use FFMPeg on SLES and RHEL, the `FFMPeg-dev` package must be installed manually.

CHAPTER

TWO

INSTALLING ROCAL WITH THE PACKAGE INSTALLER

Three rocAL packages are available:

`rocal`: The rocAL runtime package. This is the basic rocAL package that only provides dynamic libraries. It must always be installed.
`rocal-dev`: The rocAL development package. This package installs a full suite of libraries, header files, and samples. This package needs to be installed to use samples.
`rocal-test`: A test package that provides a CTest to verify the installation.

All the required prerequisites are installed when the package installation method is used.

Note

TurboJPEG must be installed manually on SLES.
To use FFMPeg on SLES and RedHat, the `FFMPeg-dev` package must be installed manually.

2.1 Basic installation

Use the following commands to install only the rocAL runtime package:

Ubuntu

```
sudo apt install rocal
```

RHEL

```
sudo yum install rocal
```

SLES

```
sudo zypper install rocal
```

2.2 Complete installation

Use the following commands to install `rocal`, `rocal-dev`, and `rocal-test`:

Ubuntu

```
sudo apt-get install rocal rocal-dev rocal-test
```

RHEL

```
sudo yum install rocal rocal-devel rocal-test
```

SLES

sudo zypper install rocal rocal-devel rocal-test

The rocAL test package will install a CTest module. Use the following steps to test the installation:

```
mkdir rocAL-test
cd rocAL-test
cmake /opt/rocm/share/rocal/test/
ctest -VV
```

CHAPTER

THREE

BUILDING AND INSTALLING ROCAL FROM SOURCE CODE

Before building and installing rocAL, ensure ROCm has been installed with the AMDGPU installer and the `rocm` usecase.

The rocAL source code is available from https://github.com/ROCm/rocAL. Use the rocAL version that corresponds to the installed version of ROCm.

rocAL supports both the HIP and OpenCL backends.

rocAL is installed in the ROCm installation directory by default. If rocAL for both HIP and OpenCL backends will be installed on the system, each version must be installed in its own custom directory and not in the default directory.

`rocAL_pybind` is not supported on the OpenCL backend.

You can choose to use the `rocAL-setup.py` setup script to install most *prerequisites*

Note

TurboJPEG must be installed manually on SLES.
To use FFmpeg on SLES and RedHat, the `FFmpeg-dev` package must be installed manually.

To build and install rocAL for the HIP backend, create the `build_hip` directory under the `rocAL` root directory. Change directory to `build_hip`:

```
mkdir build-hip
cd build-hip
```

Use `cmake` to generate a makefile:

```
cmake ../
```

If rocAL will be built for both the HIP and OpenCL backends, use the `-DCMAKE_INSTALL_PREFIX` CMake directive to set the installation directory. For example:

```
cmake -DCMAKE_INSTALL_PREFIX=/opt/hip_backend/
```

Run make:

```
make
```

Run `cmake` again to generate Python bindings for `rocal_pybind` then install:

```
sudo cmake --build . --target PyPackageInstall
sudo make install
```

The instructions to install rocAL for the OpenCL backend are similar to those for the HIP backend. Because OpenCL doesn't support `rocal_pybind`, the second `cmake` command is omitted:

```
mkdir build-ocl
cd build-ocl
cmake -DBACKEND=OPENCL ../
make
sudo make install
```

After the installation, the rocAL files will be installed under `/opt/rocm/` unless `-DCMAKE_INSTALL_PREFIX` was specified. If `-DCMAKE_INSTALL_PREFIX` was specified, the rocAL files will be installed under the specified directory.

To make and run the tests, use `make test`.

CHAPTER

FOUR

ROCAL OVERVIEW

The performance of Deep Learning applications depends upon the efficiency of performance pipelines that can load and preprocess data efficiently to provide a high throughput. The pipelines are typically used to perform tasks such as loading and decoding data, perform a variety of augmentations, perform color-format conversions, etc., before passing the data for training or inference. The Deep Learning frameworks also require the pipelines to support multiple data formats and augmentations to adapt to a variety of datasets and models. This can be achieved by creating processing pipelines that fully utilize the underlying hardware capabilities.

ROCm™ Augmentation Library (rocAL™) lets the user create hybrid pipelines to maximize the throughput for Machine Learning applications. It helps to create pipelines that can efficiently process images, videos, and a variety of storage formats. The user can program these pipelines using C or Python API. rocAL significantly accelerates data processing on AMD processors.

To optimize the preprocessing pipeline, rocAL utilizes the following features:

- Prefetching: Loads the data for the next batch while the existing batch is under process. This parallelization allows more batches to be processed in less time.
- Hybrid execution: Utilizes both the CPU and GPU simultaneously. For example, decoding the data on the CPU while running the training on the GPU.
- Hardware decoding: rocDecode and rocJPEG are used to decode data on hardware.
- Batch processing: Groups and processes the data together as a batch.

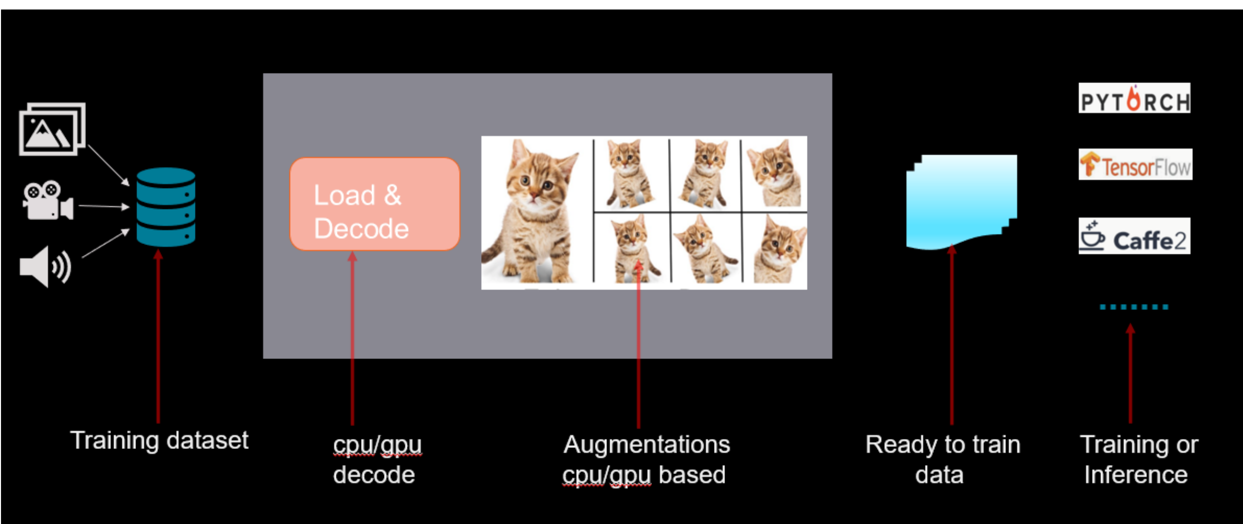

Fig. 4.1: The Role of Pipelines in Deep Learning Applications

4.1 Key Components

- CPU- or GPU-based implementation for each augmentation and data_loader nodes
- Python and C APIs for easy integration and testing
- Multiple framework support and portable on PyTorch, TensorFlow, and MXNet
- Flexible graphs to help the user create custom pipelines
- Multicore host and multi-gpu execution for the graph
- Support for various augmentations such as fish-eye, water, gitter, non-linear blend, etc., using the AMD ROCm Performance Primitive (RPP) library
- Support for classification, object detection, segmentation, and keypoint data pipelines

4.2 Third-party Integration

rocAL provides support for many operators. The module imports are designed like other available data loaders for a smooth integration with training frameworks. The rocal_pybind package provides support for integrating with PyTorch, TensorFlow, and Caffe2. rocAL also supports many data formats such as FileReader, COCO Reader, TFRecordReader, and Lightning Memory-Mapped Database (LMDB), thus offering a unified approach to framework integration.

4.3 rocAL Operators

rocAL operators offer the flexibility to run on CPU or GPU for building hybrid pipelines. They also support classification and object detection on the workload. Some of the useful operators supported by rocAL are listed below:

- **Augmentations:** These are used to enhance the data set by adding effects to the original images. To use the augmentations, import the instance of `amd.rocal.fn` into the Python script. These augmentation APIs further call the RPP kernels underneath (HIP/HOST) depending on the backend used to build RPP and rocAL.
- **Readers:** These are used to read and understand the different types of datasets and their metadata. Some examples of readers are list of files with folders, LMDB, TFRecord, and JSON file for metadata. To use the readers, import the instance of `amd.rocal.readers` into the Python script.
- **Decoders:** These are used to support different input formats of images and videos. Decoders extract data from the datasets that are in compressed formats such as JPEG, MP4, etc. To use the decoders, import the instance of `amd.rocal.decoders` into the Python script.

4.3.1 Table 1. Augmentations Available through rocAL

Color Augmentations	Effects Augmentations	Geometry Augmentations
Blend	Fog	Crop
Blur	Jitter	Crop Mirror Normalization
Brightness	Pixelization	Crop Resize
Color Temperature	Raindrops	Fisheye Lens
Color Twist	Snowflakes	Flip (Horizontal, Vertical, and Both)
Contrast	Salt and Pepper Noise	Lens Correction
Exposure		Random Crop
Gamma		Resize
Hue		Resize Crop Mirror
Saturation		Rotation
Vignette		Warp Affine

4.3.2 Table 2. Readers Available through rocAL

Readers	Description
File Reader	Reads images from a list of files in a folder(s)
Video Reader	Reads videos from a list of files in a folder(s)
Caffe LMDB Reader	Reads (key, value) pairs from Caffe LMDB
Caffe2 LMDB Reader	Reads (key, value) pairs from Caffe2 LMDB
COCO Reader - file source and keypoints	Reads images and JSON annotations from COCO dataset
TFRecord Reader	Reads from a TFRecord dataset
MXNet Reader	Reads from a RecordIO dataset
Web Dataset Reader	Reads from a web dataset
CIFAR-10 Dataset Reader	Reads from a binary CIFAR-10 dataset

4.3.3 Table 3. Decoders Available through rocAL

Decoders	Description
Image	Decodes JPEG images
Image_raw	Decodes images in raw format
Image_random_crop	Decodes and randomly crops JPEG images
Image_slice	Decodes and slices JPEG images

To see examples demonstrating the usage of decoders and readers, see rocAL Python Examples.

CHAPTER

FIVE

ROCAL ARCHITECTURE COMPONENTS

The rocAL architecture has rocAL Master-Graph and ROCm Performance Primitive (RPP) as its major components.

5.1 rocAL Master-Graph

The rocAL pipeline is built on top of rocAL Master-Graph. The architectural components of rocAL Master-Graph are described below:

Loader and Processing Modules: The rocAL Master-Graph consists of two main architectural components, a loader module to load data and a processing module to process data. The loader module is clearly separated from the processing module for a seamless execution without any blockages. The Prefetch queue helps to load data ahead of time and can be configured with user-defined parameters. The Output routine runs in parallel with the load routine, as both have separate queues for storing the result.

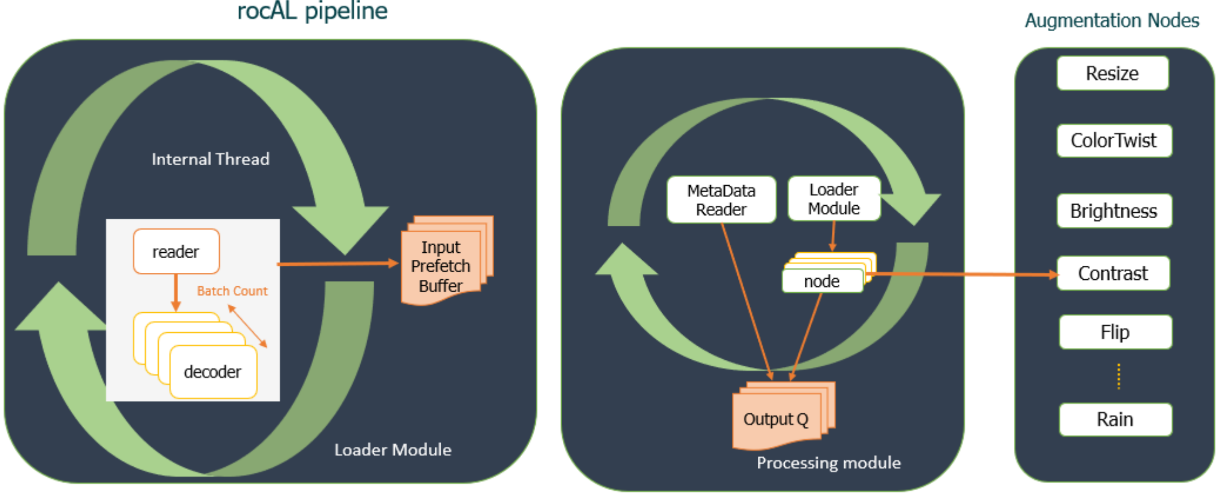

Fig. 5.1: rocAL Master-Graph Architecture

rocAL Pipeline: The rocAL pipeline holds great significance, as it contains all the information required to create a rocAL graph with data loader, augmentation nodes, and the output format. Once a rocAL pipeline is created, the user can build, run, and call an iterator to get the next batch of data into the pipeline. The user can install the rocAL pipeline using the rocAL Python package. It supports many operators for data loading and data augmentation.

5.2 ROCm Performance Primitive (RPP) Library

RPP is a comprehensive high-performance computer vision library optimized for the AMD CPU and GPU with HIP and OpenCL backends. It is available under the AMD ROCm software platform. It provides low-level functionality for all rocAL operators for single, image, and tensor datatypes. RPP provides an extensive library for vision augmentations that includes vision functions, color augmentations, filter augmentations, geometric distortions, and a few more features.

For more information on RPP along with the list of supported kernels, see ROCm Performance Primitives.

CHAPTER

SIX

USING ROCAL WITH PYTORCH FOR TRAINING

rocAL improves machine learning (ML) pipeline efficiency by preprocessing data and parallelizing data loading.

PyTorch iterators and readers are provided as plugins to separate data loading from training.

You'll need a rocAL PyTorch Docker container to run PyTorch training with rocAL.

To use rocAL with PyTorch, import the rocAL PyTorch plugin:

```
from amd.rocal.plugin.pytorch import ROCALClassificationIterator
```

Set up a training pipeline that reads data from a dataset using `readers.file` and uses `decoders.image_slice` to decode the raw images.

Call the training pipeline using `ROCALClassificationIterator`.

Two examples of PyTorch training using rocAL are available in the rocAL GitHub repository.

CHAPTER SEVEN

USING ROCAL WITH TENSORFLOW FOR TRAINING

rocAL improves machine learning (ML) pipeline efficiency by preprocessing data and parallelizing data loading.

TensorFlow iterators and readers are provided as plugins to separate data loading from training.

You'll need a rocAL TensorFlow Docker container to run TensorFlow training with rocAL.

To use rocAL with TernsorFlow, import the rocAL TensorFlor plugin:

```
from amd.rocal.plugin.tf import ROCALIterator
```

Set up a training pipeline that reads data with `readers.tfrecord` and uses `decoders.image` to decode the raw images.

Call the training pipeline using `ROCALIterator`.

An example of TensorFlow training using rocAL is available in the rocAL GitHub repository.

CHAPTER

EIGHT

USING ROCAL WITH JAX FOR TRAINING

rocAL improves machine learning (ML) pipeline efficiency by preprocessing data and parallelizing data loading.

JAX iterators are provided as plugins to separate data loading from training.

You'll need a rocAL JAX Docker container to run JAX training with rocAL.

To use rocAL with JAX, import the rocAL JAX plugin:

```
from amd.rocal.plugin.jax import ROCALJaxIterator
```

Get the number of available devices using `jax.device_count()`. Set up a training pipeline that partitions the training data and run the pipeline on each device using `ROCALJaxIterator`.

A Jupyter Notebook is available as an example of using JAX with rocAL.

CHAPTER

NINE

ROCAL EXAMPLES

Use the links below to access rocAL examples:

- Image Processing
- Pytorch
- Tensorflow
- Jupyter Notebooks

CHAPTER

TEN

ROCAL RNNT DATALOADING IN PYTHON

rocAL supports the RNNT speech recognition model through audio readers and other functions that can be used with PyTorch.

All the functions used for RNNT dataloading are available in the `amd.rocal.fn` module. See *Using rocAL with the Python API* for more details about this module.

All the augmentations used in the RNNT dataloader pipeline are available as part of rocAL. These augmentations need to be plugged into the rocAL PyTorch dataloader to run the training. PyTorch samples can be found in the rocAL GitHub repository.

Note

The rocAL GitHub repository does not host the entire RNNT dataloader source.

Table 10.1: Supported augmentations

Function	Description	Details
fn. resampl	Resamples an audio signal.	Resampling is achieved by applying a sinc filter with a Hann window. The extent is controlled by the function's `quality` argument.
fn. nonsilen	Detects leading and trailing silences.	Returns the beginning and length of the non-silent region. Compares the short-term power calculated for the window length of the signal with a silence cut-off threshold. The signal is considered to be silent when the short term power in decibels is less than the cut-off threshold in decibels.
fn. slice	Slices the input.	The slice is specified by an anchor and a shape for the slice.
fn. preempha	Applies a preemphasis filter to the input.	The filter used is `Output[t] = Input[t] - coeff * Input[t-1]` if `t > 1` and `Output[t] = Input[t] - coeff * Input_border` if `t == 0`
fn. spectrog	Produces a spectrogram from a 1D audio signal.	
fn. mel_fil	Converts a spectrogram to a mel spectrogram.	Conversion is done by applying a bank of triangular filters where the frequency dimension is selected from the input layout.
fn. to_decil	Converts a magnitude to decibels.	The conversion is done using `out[i] = multiplier * log10( max(min_ratio, input[i]/reference) )` where `min_ratio = pow(10, cutoff_db/multiplier)`.
fn. normali	Normalizes an input.	Normalization is done by removing the mean and dividing by the standard deviation.

CHAPTER

ELEVEN

USING ROCAL WITH C++ API

This chapter explains how to create a pipeline and add augmentations using C++ APIs directly. The Python APIs also call these C++ APIs internally using the Python pybind utility as explained in the section Installing rocAL Python Package.

11.1 C++ Common APIs

The following sections list the commonly used C++ APIs.

11.1.1 rocalCreate

Use: To create the pipeline

Returns: The context for the pipeline

Arguments:

- RocalProcessMode: Defines whether rocal data loading should be on the CPU or GPU.

```
RocalProcessMode:: ROCAL_PROCESS_GPU
RocalProcessMode::ROCAL_PROCESS_CPU
```

- RocalTensorOutputType: Defines whether the output of rocal tensor is FP32 or FP16.

```
RocalTensorOutputType::ROCAL_FP32
RocalTensorOutputType::ROCAL_FP16
```

See rocalCreate example.

```
extern "C"  RocalContext  ROCAL_API_CALL rocalCreate(size_t batch_size, RocalProcessMode␣
↪affinity, int gpu_id = 0, size_t cpu_thread_count = 1, size_t prefetch_queue_depth = 3,
↪ RocalTensorOutputType output_tensor_data_type = RocalTensorOutputType::ROCAL_FP32);
```

11.1.2 rocalVerify

Use: To verify the graph for all the inputs and outputs

Returns: A status code indicating the success or failure

See rocalVerify example.

```
extern "C"  RocalStatus ROCAL_API_CALL rocalVerify(RocalContext context);
```

11.1.3 rocalRun

Use: To process and run the built and verified graph

Returns: A status code indicating the success or failure

See rocalRun example.

```
extern "C"  RocalStatus  ROCAL_API_CALL rocalRun(RocalContext context);
```

11.1.4 rocalRelease

Use: To free all the resources allocated during the graph creation process

Returns: A status code indicating the success or failure

See rocalRelease example.

```
extern "C"  RocalStatus  ROCAL_API_CALL rocalRelease(RocalContext rocal_context);
```

11.1.5 Image Augmentation Using C++ API

The example below shows how to create a pipeline, read JPEG images, perform certain augmentations on them, and show the output using OpenCV by utilizing C++ API.

Listing 11.1: Example Image Augmentation

```
Auto handle = rocalCreate(inputBatchSize, processing_device?RocalProcessMode::ROCAL_
↪PROCESS_GPU:RocalProcessMode::ROCAL_PROCESS_CPU, 0,1);
input1 = rocalJpegFileSource(handle, folderPath1,  color_format, shard_count, false,␣
↪shuffle, false,  ROCAL_USE_USER_GIVEN_SIZE, decode_width, decode_height, dec_type);

image0 = rocalResize(handle, input1, resize_w, resize_h, true);

RocalImage image1 = rocalRain(handle, image0, false);


    RocalImage image11 = rocalFishEye(handle, image1, false);


    rocalRotate(handle, image11, true, rand_angle);


    // Creating successive blur nodes to simulate a deep branch of augmentations
    RocalImage image2 = rocalCropResize(handle, image0, resize_w, resize_h, false, rand_
↪crop_area);;
    for(int i = 0 ; i < aug_depth; i++)
    {
        image2 = rocalBlur(handle, image2, (i == (aug_depth -1)) ? true:false );
    }
// Calling the API to verify and build the augmentation graph
    if(rocalVerify(handle) != ROCAL_OK)
    {
        std::cout << "Could not verify the augmentation graph" << std::endl;
        return -1;
    }
```

(continues on next page)

(continued from previous page)

```
while (!rocalIsEmpty(handle))
    {
        if(rocalRun(handle) != 0)
            break;
}
```

To see a sample image augmentation application in C++, see Image Augmentation.

CHAPTER

TWELVE

ROCAL C++ API

Consult the following files for detailed information on the rocAL C++ API:

- Functions
- Data types
- Augmentations
- Data loaders
- Data transfer
- Info
- Metadata
- Parameters

CHAPTER THIRTEEN

ROCAL PYTHON API OVERVIEW

The rocAL Python package has been created using Pybind11 which enables data transfer between the rocAL C++ API and Python API. The `rocal_pybind` package includes both PyTorch and TensorFlow framework support and support for multiple data readers such as `FileReader`, `COCOReader`, and `TFRecordReader`.

The rocAL data types are defined in amd.rocal.types.

`amd.rocal.fn`
: Contains the image augmentations linked to the rocAL C++ API.

`amd.rocal.decoders`
: Image, video, and audio decoders.

`amd.rocal.readers`
: Image, video, and audio readers.

`amd.rocal.pipeline`
: The pipeline class encapsulates the data needed to build and run a rocAL graph. This includes support for context and graph creation, functions to verify and run the graph, and data transfer functions.

`amd.rocal.types`
: enums exported from the C++ API to Python.

`amd.rocal.plugin.pytorch`
: PyTorch plugin that includes the `ROCALGenericIterator` for Pytorch. The `ROCALClassificationIterator` class implements an iterator for image classification that returns labelled images.

`amd.rocal.plugin.tf`
: TensorFlow plugin that includes the `readers.tfrecord` TensorFlow reader.

`amd.rocal.plugin.jax`
: JAX plugin that includes the `ROCALJaxIterator` for JAX. The `ROCALJaxIterator` implements an iterator for running a pipeline over multiple devices.

CHAPTER

FOURTEEN

ROCAL PYTHON API

Consult the following files for detailed information on the rocAL Python API:

- Readers
- Decoders
- Augmentations
- Pipeline
- Randomization
- PyTorch integration
- TensorFlow integration
- JAX integration

CHAPTER FIFTEEN

LICENSE

MIT License

Copyright (c) 2022 - 2025 Advanced Micro Devices, Inc. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.