DALI

DU-09050-001 _v0.1 Beta Release | July 2018

Developer Guide

TABLE OF CONTENTS

Chapter 1. WHAT IS DALI?

DALI is a data loading library that accelerates the preprocessing of input data for deep learning applications. By offloading augmentations onto GPUs, DALI addresses performance bottlenecks in today's computer vision deep learning applications that include complex, multi-stage data augmentation steps. With DALI 0.1 beta release, deep learning researchers can scale training performance on image classification models such as ResNet-50 with MXNet™ , TensorFlow™ , and PyTorch™ across Amazon Web Services P3 8 GPU instances or DGX-1 systems with Volta architecture. Framework developers will have less duplication due to better code reuse and maintainability.

DALI offers both performance and flexibility of accelerating different data pipelines (graphs that can have multiple inputs and outputs), as a single library, that can be easily integrated into different deep learning training and inference applications.

1.1. Benefits Of DALI

There are 3 key factors that DALI brings to deep learning training and inference applications:

Performance

On dense GPU systems, deep learning applications can be significantly bottlenecked on the CPU, limiting the overall performance and scalability of training and inference tasks. DALI enables offloading key deep learning augmentation steps on to GPUs, alleviating CPU bottleneck on the deep learning preprocessing pipelines. This results in out-of-box performance of overall training workflow and efficient utilization of multi-GPU resources on the system.

Drop-in Integration

DALI comes with built-in plugins for key frameworks such as MXNet, TensorFlow, and PyTorch™ . This enables automatic integration with frameworks so that researchers and developers can get up and running with DALI easily and quickly.

Flexibility

DALI supports multiple input data formats that are commonly used in computer vision deep learning applications, for example, JPEG images, raw formats, Lightning Memory-Mapped Database (LMDB), RecordIO and TFRecord. The flexibility of input data formats allows portability of training workflows across different frameworks and models, and helps to avoid intermediate data conversion steps. DALI enables better code reuse and maintainability with optimized building blocks and support for different data formats.

1.2. Where Does DALI Fit?

DALI focuses on data loading and augmentations, in other words, all the preprocessing stages before you start training your model.

Without DALI, a ResNet-50 model pipeline operations are primarily processed on CPUs. These functions are implemented differently in each of the frameworks and not currently optimized to scale across multi-GPU environments.

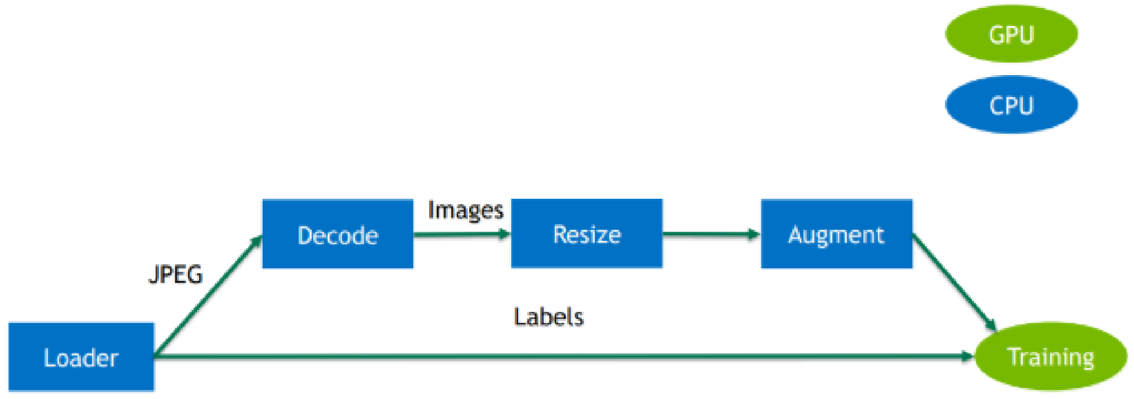

Figure 1 ResNet-50 pipeline without DALI

With DALI, the same pipeline is now accelerated by offloading key augmentation functions onto the GPU. The hybrid approach of efficiently utilizing available CPU and GPU resources helps maximize the overall training and inference performance.

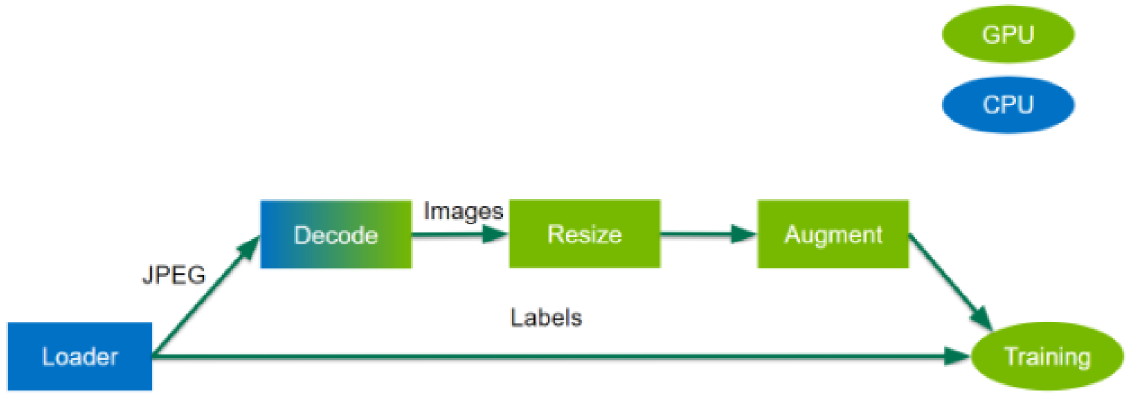

Figure 2 ResNet-50 pipeline with DALI

1.3. How Do I Get DALI?

For step-by-step instructions on how to install DALI, see the DALI Quick Start Guide.

Chapter 2. GETTING STARTED WITH DALI

The following tasks assume you've already installed DALI. If you have not installed DALI, see the DALI Quick Start Guide.

The following sections highlight the user goals and tasks that you can perform with DALI. In DALI 0.1 beta release, you can define and execute a pipeline graph. In order to perform these tasks, ensure you have the following software installed:

- Optionally, install Jupyter to work in interactive mode.
- Install NumPy.
- Install Matplotlib.

To interact with the code, see the Getting Started Tutorial.

2.1. Defining A Pipeline Graph

DALI data pipelines are graphs that can have multiple outputs and inputs. Fundamentally, a pipeline can have multiple data processing connections where the output of one connection is the input of the next connection. The pipeline class contains all the necessary information and multiple functions related to defining, building and running the pipeline.

In order to create our own input and augmentation pipeline, we will make subclasses of it. The first step to running your data pipeline is to import DALI.

```
from dali.pipeline import Pipeline
```

Let's define a simple pipeline for a classifier that determines whether a picture contains a cat or dog. Our pipeline is called **SimplePipeline**.

```
import os
import fnmatch

for root, dir, files in os.walk("images"):
        depth = root.count('/')
        ret = ""
        if depth > 0:
            ret += "  " * (depth - 1) + "|-"
        print ret + root
        for items in fnmatch.filter(files, "*"):
                print (" " * len(ret)) + "|-" + items
```

We prepared a directory structure that contains pictures of dogs and cats. The following output code shows the structure of our directory:

```
images
|-images/dog
  |-dog_1.jpg
  |-dog_8.jpg
  |-dog_7.jpg
  |-dog_6.jpg
  |-dog_2.jpg
  |-dog_11.jpg
  |-dog_10.jpg
  |-dog_9.jpg
  |-dog_5.jpg
  |-dog_4.jpg
  |-dog_3.jpg
|-images/kitten
  |-cat_7.jpg
  |-cat_10.jpg
  |-cat_3.jpg
  |-cat_2.jpg
  |-cat_4.jpg
  |-cat_9.jpg
  |-cat_1.jpg
  |-cat_8.jpg
  |-cat_5.jpg
  |-cat_6.jpg
```

This pipeline will read images from the directory where the images are stored, decode them, and return (image, label) pairs.

```
import dali.ops as ops
import dali.types as types

image_dir = "images"
batch_size = 8

class SimplePipeline(Pipeline):
    def __init__(self, batch_size, num_threads, device_id):
        super(SimplePipeline, self).__init__(batch_size, num_threads, device_id,
 seed = 12)
        self.input = ops.FileReader(file_root = image_dir)
        self.decode = ops.HostDecoder(output_type = types.RGB)

    def define_graph(self):
        jpegs, labels = self.input()
        images = self.decode(jpegs)
        return (images, labels)
```

The **SimplePipeline** class is a subclass of **dali.pipeline.Pipeline**, which provides most of the methods to create and launch a pipeline. The only two methods we need to implement is the constructor, (**__init__**), and **define_graph** function.

Constructor Function

In the constructor function, we first call our superclass constructor, in order to set global parameters of the pipeline. The global parameters consist of:

- batch size
- number of threads used to perform computation on the CPU
- which GPU device to use (**`SimplePipeline`** does not yet use GPU for compute though)
- seed for random number generation

We also define member variables of our **`SimplePipeline`** class as operations defined in the **`dali.ops`** module. The member variables are:

FileReader
Traverses the directory and returns pairs of (encoded image, label).

HostDecoder
Takes an encoded image input and outputs decoded RGB image.

`define_graph` Function

In the **`define_graph`** function, we define the actual flow of computation. We use our input operation to create **`jpegs`** (encoded images) and labels.

```
jpegs, labels = self.input()
```

Next, we use the **`decode`** operation to create **`images`** (decoded RGB images).

```
images = self.decode(jpegs)
```

Finally, we specify which of the intermediate variables should be returned as outputs of the pipeline.

```
return (images, labels)
```

2.2. Building A Pipeline Graph

Before we can use our **`SimplePipeline`**, we need to build it by calling the **`build`** function.

```
pipe = SimplePipeline(batch_size, 1, 0)
pipe.build()
```

2.3. Running A Pipeline Graph

We're now ready to run our **`SimplePipeline`** and view the batch of results.

```
pipe_out = pipe.run()
print(pipe_out)
```

The output of **SimplePipeline** is saved to the **pipe_out** variable. The output is a list of two elements since we specified two outputs in the **define_graph** function in the **SimplePipeline** class. Both of these elements are **TensorListCPU** objects; meaning, each element contains a list of tensors on the CPU.

In order to show the results (for debugging purposes since during the actual training we would not do this step because it would make our batch of images do a round trip from GPU to CPU and back), we can send our data from DALI's Tensor to NumPy array. Not every **TensorList** can be accessed that way though. **TensorList** is more general than NumPy array and can hold tensors with different shapes. In order to check whether we can send it to NumPy directly, we can call the **is_dense_tensor** function of **TensorList**.

```
images, labels = pipe_out
print("Images is_dense_tensor: " + str(images.is_dense_tensor()))
print("Labels is_dense_tensor: " + str(labels.is_dense_tensor()))


Images is_dense_tensor: False
Labels is_dense_tensor: True
```

As it turns out, **TensorList** containing labels can be represented by a tensor, while the **TensorList** containing images cannot.

Let us see, what is the shape and contents of the returned labels.

```
import numpy as np

labels_tensor = labels.as_tensor()

print (labels_tensor.shape())
print (np.array(labels_tensor))
```

In order to see the images, we will need to loop over all tensors contained in **TensorList**, accessed with its **at** method.

```
from __future__ import division
import matplotlib.gridspec as gridspec
import matplotlib.pyplot as plt
%matplotlib inline

def show_images(image_batch):
    columns = 4
    rows = (batch_size + 1) // (columns)
    fig = plt.figure(figsize = (32,(32 // columns) * rows))
    gs = gridspec.GridSpec(rows, columns)
    for j in range(rows*columns):
        plt.subplot(gs[j])
        plt.axis("off")
        plt.imshow(image_batch.at(j))


show_images(images)
```

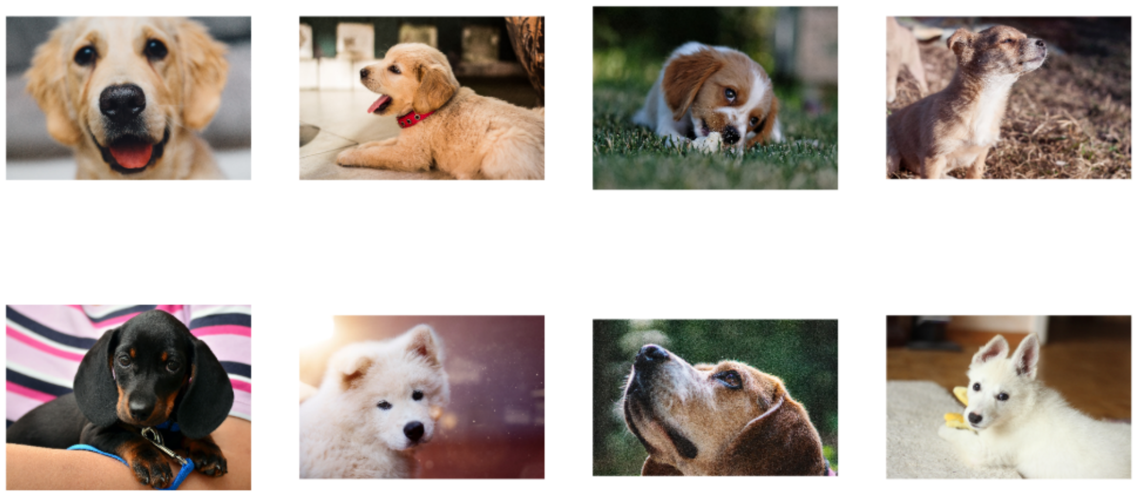

Figure 3 SimplePipeline results output

Chapter 3. SUPPORTED OPERATIONS

The following sections describe the operations that are supported by DALI. These operations enable you to create the desired input and augmentation pipeline.

3.1. Color Augmentation Operators

The color augmentation operators enable you to change the color of the image.

3.1.1. `Brightness`

The **`Brightness`** class controls the brightness of an image.

The following table lists the supported parameters for the **`Brightness`** class.

Table 1 **`Brightness`** class parameters

Parameter	Required	Description	Type	Values
`brightness`	No	Brightness change factor values >=0 are accepted.	`float` or `float tensor`	A value of `0` produces a black image. A value of `1` is no change. A value of `2` increases the light twice as much. The default value is `1.000000`.
`image_type`	No	The color space of the input and output image.	`dali.types.DALIImageType`	The default value is `RGB`.

3.1.2. `Contrast`

The **`Contrast`** class controls the color contrast of the image.

The following table lists the supported parameters for the **`Contrast`** class.

Table 2 **`Contrast`** class parameters

Parameter	Required	Description	Type	Values
`contrast`	No	Contrast change factor values >=0 are accepted.	`float` or `float tensor`	A value of `0` produces a gray image. A value of `1` is no change. A value of `2` increases the contrast twice as much. The default value is `1.000000`.
`image_type`	No	The color space of the input and output image.	`dali.types.DALIImageType`	The default value is`RGB`.

3.1.3. `Hue`

The **`Hue`** class controls the hue level of the image.

The following table lists the supported parameters for the **`Hue`** class.

Table 3 **`Hue`** class parameters

Parameter	Required	Description	Type	Values
`hue`	No	Hue change in angles.	`float` or `float tensor`	The default value is`0.000000`.
`image_type`	No	The color space of the input and output image.	`dali.types.DALIImageType`	The default value is`RGB`.

3.1.4. `Saturation`

The **`Saturation`** class controls the saturation level of the image.

The following table lists the supported parameters for the **`Saturation`** class.

Table 4 **`Saturation`** class parameters

Parameter	Required	Description	Type	Values
`image_type`	No	The color space of the input and output image.	`dali.types.DALIImageType`	The default value is`RGB`.

Parameter	Required	Description	Type	Values
`saturation`	No	Saturation change factor.	`float` or `float tensor`	Values >=0 are supported. For example: A value of `0` gives you a completely desaturated image. A value of `1` is no change to the images saturation. The default value is `1.000000`.

3.2. Decoder Operators

The decoder operators enable you to decode encoded input into an image.

3.2.1. HostDecoder

The **`HostDecoder`** class decodes images on the host using OpenCV. When applicable, it will pass execution to faster, format-specific decoders, like **`libjpeg-turbo`**. Output of the decoder is in **`HWC`** ordering.

The following table lists the supported parameters for the **`HostDecoder`** class.

Table 5 **`HostDecoder`** class parameters

Parameter	Required	Description	Type	Values
`output_type`	No	The color space of the output image.	`dali.types.DALIImageType`	The default value is`RGB`.

3.2.2. nvJPEGDecoder

The **`nvJPEGDecoder`** decodes JPEG images using the nvJPEG library. Output of the decoder is on the GPU and uses an **`HWC`** ordering.

The following table lists the supported parameters for the **`nvJPEGDecoder`** class.

Table 6 **`nvJPEGDecoder`** class parameters

Parameter	Required	Description	Type	Values
`output_type`	No	The color space of the output image.	`dali.types.DALIImageType`	The default value is`RGB`.
`use_batched_decode`	No	Use nvJPEG's batched decoding API.	`bool`	The default value is`False`.

3.3. Displacement Operators

The displacement operators enable you to perform spatial transformations (such as rotation) of images.

3.3.1. `Jitter`

The **`Jitter`** class performs a random jitter augmentation. The output image is produced by moving each pixel by a random amount bounded by half of **`nDegree`** parameter (in both **`x`** and **`y`** dimensions).

The following table lists the supported parameters for the **`Jitter`** class.

Table 7 **`Jitter`** class parameters

Parameter	Required	Description	Type	Values
`fill_value`	No	Color value used for padding pixels.	`float`	The default value is`0.000000`.
`interp_type`	No	Type of interpolation used.	`dali.types.DALIInterpType`	The default value is`INTERP_NN`.
`mask`	No	Whether to apply this augmentation to the input image or not.	`int` or `int tensor`	A value of `0` will not apply this transformation. A value of `1` will apply this transformation. The default value is`1`.
`nDegree`	No	Each pixel is moved by a random amount in range `[-nDegree/2, nDegree/2]`.	`int`	The default value is`2`.

3.3.2. `Rotate`

The **`Rotate`** class rotates the image.

The following table lists the supported parameters for the **`Rotate`** class.

Table 8 **`Rotate`** class parameters

Parameter	Required	Description	Type	Values
`angle`	Yes	Rotation angle.	`float` or `float tensor`	

Parameter	Required	Description	Type	Values
`fill_value`	No	Color value used for padding pixels.	`float`	The default value is`0.000000`.
`interp_type`	No	Type of interpolation used.	`dali.types.DALIInterpType`	The default value is `INTERP_NN`.
`mask`	No	Whether to apply this augmentation to the input image.	`int` or `int tensor`	A value of `0` will not apply this transformation. A value of `1` will apply this transformation. The default value is `1`.

3.3.3. Sphere

The **`Sphere`** class performs a sphere augmentation.

The following table lists the supported parameters for the **`Sphere`** class.

Table 9 **`Sphere`** class parameters

Parameter	Required	Description	Type	Values
`fill_value`	No	Color value used for padding pixels.	`float`	The default value is `0.000000`.
`interp_type`	No	Type of interpolation used.	`dali.types.DALIInterpType`	The default value is `INTERP_NN`.
`mask`	No	Whether to apply this augmentation to the input image.	`int` or `int tensor`	A value of `0` will not apply this transformation. A value of `1` will apply this transformation. The default value is `1`.

3.3.4. WarpAffine

The **`WarpAffine`** class applies an affine transformation to the image.

The following table lists the supported parameters for the **`WarpAffine`** class.

Table 10 **`WarpAffine`** class parameters

Parameter	Required	Description	Type	Values
`matrix`	Yes	Matrix of the transform (`dst - src`).	`list of float`	Given a list of values (`M11, M12, M13, M21,`

Parameter	Required	Description	Type	Values
				M22, M23) this operation will produce a new image using the formula: `dst(x,y) = src(M11 * x + M12 * y + M13, M21 * x + M22 * y + M23)` It is equivalent to OpenCV's **warpAffine** operation with a flag **WARP_INVERSE_MAP** set.
fill_value	No	Color value used for padding pixels.	**float**	The default value is 0.000000.
interp_type	No	Type of interpolation used.	**dali.types.DALIInterpType**	The default value is **INTERP_NN**.
mask	No	Whether to apply this augmentation to the input image.	**int** or **int tensor**	A value of 0 will not apply this transformation. A value of 1 will apply this transformation. The default value is 1.
use_image_center	No	Whether to use the image center as the center of transformation.	**bool**	When set to **true**, the coordinates are calculated from the center of the image. The default value is **False**.

3.3.5. Water

The **Water** class performs a water augmentation.

The following table lists the supported parameters for the **Water** class.

Table 11 **`Water`** class parameters

Parameter	Required	Description	Type	Values
`ampl_x`	No	Amplitude of the wave in `x` direction.	`float`	The default value is `10.000000`.
`ampl_y`	No	Amplitude of the wave in `y` direction.	`float`	The default value is `10.000000`.
`fill_value`	No	Color value used for padding pixels.	`float`	The default value is `0.000000`.
`freq_x`	No	Frequency of the wave in `x` direction.	`float`	The default value is `0.049087`.
`freq_y`	No	Frequency of the wave in `y` direction.	`float`	The default value is `0.049087`.
`interp_type`	No	Type of interpolation used.	`dali.types.DALIInterpType`	The default value is `INTERP_NN`.
`mask`	No	Whether to apply this augmentation to the input image.	`int` or `int tensor`	A value of `0` will not apply this transformation. A value of `1` will apply this transformation. The default value is `-1`.
`phase_x`	No	Phase of the wave in `x` direction.	`float`	The default value is `0.000000`.
`phase_y`	No	Phase of the wave in `y` direction.	`float`	The default value is `0.000000`.

3.4. Normalize Operators

The normalize operators enable you to normalize the images with mean and standard deviation, as well as prepare them for ingestion in the framework by converting datatype to **`float`** and layout to **`NCHW`**.

3.4.1. `CropMirrorNormalize`

The **`CropMirrorNormalize`** class performs fused cropping, normalization, format conversion (**`NHWC`** to **`NCHW`**) if desired, and type casting. The normalization takes the input image and produces an output using the formula:

```
output = (input - mean) / std
```

The following table lists the supported parameters for the **`CropMirrorNormalize`** class.

Table 12 **CropMirrorNormalize** class parameters

Parameter	Required	Description	Type	Values
crop	Yes	Size of the cropped image.	**int** or **list of int**	If only a single value of **c** is provided, the resulting crop will be a square with size (**c, c**).
mean	Yes	Mean pixel values for image normalization.	**list of float**	
std	Yes	Standard deviation values for image normalization.	**list of float**	
crop_pos_x	No	Horizontal position of the crop in image coordinates (0.0 - 1.0).	**float** or **float tensor**	The default value is **0.500000**.
crop_pos_y	No	Vertical position of the crop in image coordinates (0.0 - 1.0).	**float** or **float tensor**	The default value is **0.500000**.
image_type	No	The color space of the input and output image.	**dali.types.DALIImageType**	The default value is **RGB**.
mirror	No	Mask for horizontal flip.	**int** or **int tensor**	A value of **0** will not perform a horizontal flip for this image. A value of **1** will perform a horizontal flip for this image. The default value is **0**.
output_dtype	No	Output data type.	**dali.types.DALIDataType**	The default value is **FLOAT**.
output_layout	No	Output tensor data type.	**dali.types.DALITensorLayout**	The default value is **NCHW**.
pad_output	No	Whether to pad the output to the number of channels being multiple of 4.	**bool**	The default value is **False**.

3.4.2. NormalizePermute

The **NormalizePermute** class performs fused normalization, format conversion from **NHWC** to **NCHW** and type casting. Normalization takes an input image and produces output using the formula:

```
output = (input - mean) / std
```

The following table lists the supported parameters for the **NormalizePermute** class.

Table 13 **NormalizePermute** class parameters

Parameter	Required	Description	Type	Values
`height`	Yes	Height of the input image.	`int`	
`mean`	Yes	Mean pixel values for image normalization.	`list of float`	
`std`	Yes	Standard deviation values for image normalization.	`list of float`	
`width`	Yes	Width of the input image.	`int`	
`image_type`	No	The color space of the input and output image.	`dali.types.DALIImageType`	The default value is `RGB`.
`output_dtype`	No	Output data type.	`dali.types.DALIDataType`	The default value is `FLOAT`.

3.5. Reader Operators

The reader operators enable you to read data stored on the disk in various formats.

3.5.1. Caffe2Reader

The **Caffe2Reader** class reads the sample data from a Caffe2™ LMDB.

The following table lists the supported parameters for the **Caffe2Reader** class.

Table 14 **Caffe2Reader** class parameters

Parameter	Required	Description	Type	Values
`path`	Yes	Path to Caffe2 LMDB.	`string`	
`additional_inputs`	No	Additional auxiliary data tensors provided for each sample.	`int`	The default value is `0`.

Parameter	Required	Description	Type	Values
`bbox`	No	Denotes if bounding-box information is present.	`bool`	The default value is `False`.
`initial_fill`	No	Size of the buffer used for shuffling.	`int`	The default value is `1024`.
`label_type`	No	Enum describing the type of label stored in the dataset.	`int`	`SINGLE_LABEL = 0` is a single integer label for multi-class classification. `MULTI_LABEL_SPARSE = 1` is a sparse active label indices for multi-label classification. `MULTI_LABEL_DENSE = 2` is a dense label embedding vector for label embedding regression. `MULTI_LABEL_WEIGHTED_SPARSE = 3` is a sparse active label indices with per-label weights for multi-label classification. The default value is `0`.
`num_labels`	No This parameter is required when sparse labels are used.	Number of classes in the dataset.	`int`	The default value is `1`.
`num_shards`	No	Partitions the data into this many parts.	`int`	The default value is `1`.
`random_shuffle`	No	Whether to shuffle data or not.	`bool`	The default value is `False`.
`shard_id`	No	The id of the part to read.	`int`	The default value is `0`.
`tensor_init_bytes`	No	Hint for how much memory to allocate per image.	`int`	The default value is `1048576`.

3.5.2. `CaffeReader`

The **CaffeReader** class reads (image and label) pairs from a Caffe™ LMDB.

The following table lists the supported parameters for the **CaffeReader** class.

Table 15 **CaffeReader** class parameters

Parameter	Required	Description	Type	Values
path	Yes	Path to Caffe LMDB.	**string**	
initial_fill	No	Size of the buffer used for shuffling.	**int**	The default value is **1024**.
num_shards	No	Partitions the data into this many parts.	**int**	The default value is **1**.
random_shuffle	No	Whether to shuffle data or not.	**bool**	The default value is **False**.
shard_id	No	The id of the part to read.	**int**	The default value is **0**.
tensor_init_bytes	No	Hint for how much memory to allocate per image.	**int**	The default value is **1048576**.

3.5.3. FileReader

The **FileReader** class reads (image and label) pairs from a directory.

The following table lists the supported parameters for the **FileReader** class.

Table 16 **FileReader** class parameters

Parameter	Required	Description	Type	Values
file_root	Yes	Path to a directory containing data files.	**string**	
file_list	No	Path to the file with a list of pairs **file label**.	**string**	Leave empty to traverse the **file_root** directory to obtain files and labels.
initial_fill	No	Size of the buffer used for shuffling.	**int**	The default value is **1024**.
num_shards	No	Partitions the data into this many parts.	**int**	The default value is **1**.
random_shuffle	No	Whether to shuffle data or not.	**bool**	The default value is **False**.
shard_id	No	The id of the part to read.	**int**	The default value is **0**.

Parameter	Required	Description	Type	Values
`tensor_init_bytes`	No	Hint for how much memory to allocate per image.	**`int`**	The default value is **`1048576`**.

3.5.4. **`MXNetReader`**

The **`MXNetReader`** class reads sample data from an MXNet RecordIO.

The following table lists the supported parameters for the **`MXNetReader`** class.

Table 17 **`MXNetReader`** class parameters

Parameter	Required	Description	Type	Values
`index_path`	Yes	List (of length `1`) containing a path to index (**`.idx`**) file. It is generated by the MXNet **`im2rec.py`** script together with an RecordIO file. It can also be generated using the **`rec2idx`** script distributed with DALI.	**`list of string`**	
`path`	Yes	List of paths to RecordIO files.	**`list of string`**	
`initial_fill`	No	Size of the buffer used for shuffling.	**`int`**	The default value is **`1024`**.
`num_shards`	No	Partitions the data into this many parts.	**`int`**	The default value is **`1`**.
`random_shuffle`	No	Whether to shuffle data or not.	**`bool`**	The default value is **`False`**.
`shard_id`	No	The id of the part to read.	**`int`**	The default value is **`0`**.
`tensor_init_bytes`	No	Hint for how much memory to allocate per image.	**`int`**	The default value is **`1048576`**.

3.5.5. **`TFRecordReader`**

The **`TFRecordReader`** class reads sample data from a TensorFlow TFRecord file.

The following table lists the supported parameters for the **`TFRecordReader`** class.

Table 18 **`TFRecordReader`** class parameters

Parameter	Required	Description	Type	Values
`features`	Yes	Dictionary of names and configuration of features existing in the TFRecord file. Typically obtained using helper functions `dali.tfrecord.FixedLenFeature` and `dali.tfrecord.VarLenFeature`, they are equivalent to TensorFlow `tf.FixedLenFeature` and `tf.VarLenFeature` functions respectively.	`dict of (string, dali.tfrecord.Feature)`	
`index_path`	Yes	List of paths to index files (one index file for every TFRecord file). Index files may be obtained from the TFRecord files using the `tfrecord2idx` script distributed with DALI.	`list of string`	
`path`	Yes	List of paths to TFRecord files.	`list of string`	
`initial_fill`	No	Size of the buffer used for shuffling.	`int`	The default value is `1024`.
`num_shards`	No	Partitions the data into this many parts.	`int`	The default value is `1`.
`random_shuffle`	No	Whether to shuffle data or not.	`bool`	The default value is `False`.
`shard_id`	No	The id of the part to read.	`int`	The default value is `0`.
`tensor_init_bytes`	No	Hint for how much memory to allocate per image.	`int`	The default value is `1048576`.

3.6. Resize Operators

The resize operators enable you to resize images.

3.6.1. `FastResizeCropMirror`

The **`FastResizeCropMirror`** class performs a fused resize, crop, and mirror operation. It handles both fixed and random resizing and cropping. It also backprojects the desired crop through the resize operation to reduce the amount of work performed.

The following table lists the supported parameters for the **`FastResizeCropMirror`** class.

Table 19 **`FastResizeCropMirror`** class parameters

Parameter	Required	Description	Type	Values
`crop`	Yes	Size of the cropped image.	`int` or `list of int`	If only a single value of `c` is provided, the resulting crop will be a square with size (`c,c`).
`crop_pos_x`	No	Horizontal position of the crop in image coordinates (0.0 - 1.0).	`float` or `float tensor`	The default value is `0.500000`.
`crop_pos_y`	No	Vertical position of the crop in image coordinates (0.0 - 1.0).	`float` or `float tensor`	The default value is `0.500000`.
`mirror`	No	Mask for horizontal flip.	`int` or `int tensor`	A value of `0` will not perform a horizontal flip for this image. A value of `1` will perform a horizontal flip for this image. The default value is `0`.
`resize_shorter`	No	The length of the shorter dimension of the resized image. This option is mutually exclusive with `resize_x` and `resize_y`. The operation will keep the aspect ratio of the original image.	`float` or `float tensor`	The default value is `0.000000`.
`resize_x`	No	The length of the `x` dimension of the resized image. This	`float` or `float tensor`	The default value is `0.000000`.

Parameter	Required	Description	Type	Values
		option is mutually exclusive with **`resize_shorter`**. If the **`resize_y`** is left at `0`, then the operation will keep the aspect ratio of the original image.		
`resize_y`	No	The length of the **`Y`** dimension of the resized image. This option is mutually exclusive with **`resize_shorter`**. If the **`resize_x`** is left at `0`, then the operation will keep the aspect ratio of the original image.	**`float`** or **`float tensor`**	The default value is `0.000000`.

3.6.2. RandomResizedCrop

The **`RandomResizedCrop`** class performs a crop with a randomly chosen area and aspect ratio, then resizes it to a given size.

The following table lists the supported parameters for the **`RandomResizedCrop`** class.

Table 20 **`RandomResizedCrop`** class parameters

Parameter	Required	Description	Type	Values
`size`	Yes	Size of the resized image.	**`list of float`**	
`interp_type`	No	Type of interpolation used.	**`dali.types.DALIInterpType`**	The default value is **`INTERP_LINEAR`**.
`num_attempts`	No	Maximum number of attempts used to choose random area and aspect ratio. If the maximum number of attempts is reached without finding the crop that fits in the input image, then a square shaped crop from the center of the image is chosen instead.	**`int`**	The default value is `10`.

Parameter	Required	Description	Type	Values
		The square size is equal to the shorter side of the input image.		
`random_area`	No	Range from which to choose a random area factor `A`. Before resizing, the cropped images area will be equal to `A * original` images area.	`list of float`	The default value is `[0.080000, 1.000000, ]`.
`random_aspect_ratio`	No	Range from which to choose the random aspect ratio.	`list of float`	The default value is `[0.750000, 1.333333, ]`.

3.6.3. Resize

The **`Resize`** class resizes images. This class controls both fixed and random resizes, along with fuse cropping (random and fixed), and image mirroring.

The following table lists the supported parameters for the **`Resize`** class.

Table 21 **`Resize`** class parameters

Parameter	Required	Description	Type	Values
`resize_a`	Yes	Lower bound for resize.	`int`	If neither `random_resize` nor `warp_resize` is set, then the shorter side of the input image is resized to `resize_a` and `resize_b` is ignored. If `warp_image` is set and `random_resize` is not set, then the input image is resized so that the height is `resize_a` and the width is `resize_b`. If `random_resize` is set and `warp_resize` is not set, then the shorter side

Parameter	Required	Description	Type	Values
				of the input image is resized to a random value between **`[resize_a, resize_b]`**. If both **`random_resize`** and **`warp_resize`** are set, then both sides of the input image are resized to random values in range **`[resize_a, resize_b]`**.
`resize_b`	Yes	Upper bound for resize.	**`int`**	If neither **`random_resize`** nor **`warp_resize`** is set, then this parameter is ignored. If **`warp_image`** is set and **`random_resize`** is not set, then the input image is resized so that the height is **`resize_a`** and the width is **`resize_b`**. If **`random_resize`** is set and **`warp_resize`** is not set, then the shorter side of the input image is resized to a random value between **`[resize_a, resize_b]`**. If both **`random_resize`** and **`warp_resize`** are set, then both sides of the input image are resized to random values in range **`[resize_a, resize_b]`**.

Parameter	Required	Description	Type	Values
`image_type`	No	The color space of the input and output image.	`dali.types.DALIImageType`	The default value is `RGB`.
`interp_type`	No	Type of interpolation used.	`dali.types.DALIInterpType`	The default value is `INTERP_LINEAR`
`random_resize`	No	Whether to randomly resize images or not.	`bool`	The default value is `False`.
`save_attrs`	No	Save the reshape attributes for testing.	`bool`	The default value is `False`.
`warp_resize`	No	Whether to modify the aspect ratio of the image.	`bool`	The default value is `False`.

3.6.4. `ResizeCropMirror`

The **`ResizeCropMirror`** class performs a fused resize, crop, and mirror operation. It handles both fixed and random resizing and cropping.

The following table lists the supported parameters for the **`ResizeCropMirror`** class.

Table 22 **`ResizeCropMirror`** class parameters

Parameter	Required	Description	Type	Values
`crop`	Yes	Size of the cropped image.	`int` or `list of int`	If only a single value of `c` is provided, the resulting crop will be a square with size (`c,c`).
`crop_pos_x`	No	Horizontal position of the crop in image coordinates (0.0 - 1.0).	`float` or `float tensor`	The default value is `0.500000`.
`crop_pos_y`	No	Vertical position of the crop in image coordinates (0.0 - 1.0).	`float` or `float tensor`	The default value is `0.500000`.
`mirror`	No	Mask for horizontal flip.	`int` or `int tensor`	A value of `0` will not perform a horizontal flip for this image. A value of `1` will perform a horizontal flip for this image. The default value is `0`.

Parameter	Required	Description	Type	Values
`resize_shorter`	No	The length of the shorter dimension of the resized image. This option is mutually exclusive with **`resize_x`** and **`resize_y`**. The operation will keep the aspect ratio of the original image.	**`float`**	The default value is `0.000000`.
`resize_x`	No	The length of the **`X`** dimension of the resized image. This option is mutually exclusive with **`resize_shorter`**. If the **`resize_y`** is left at `0`, then the operation will keep the aspect ratio of the original image.	**`float`**	The default value is `0.000000`.
`resize_y`	No	The length of the **`Y`** dimension of the resized image. This option is mutually exclusive with **`resize_shorter`**. If the **`resize_x`** is left at `0`, then the operation will keep the aspect ratio of the original image.	**`float`**	The default value is `0.000000`.

3.7. Support Operators

The support operators are a class of operators, the result of which can be used as arguments to other functions. Currently only Random Number Generators are members of this class.

3.7.1. `CoinFlip`

The **`CoinFlip`** class produces tensor filled with **`0`** and **`1`**; the results of a random coin flip. It's useable as an argument for select operations.

The following table lists the supported parameters for the **`CoinFlip`** class.

Table 23 **CoinFlip** class parameters

Parameter	Required	Description	Type	Values
probability	No	Probability of returning 1.	**float**	The default value is 0.500000.

3.7.2. Uniform

The **Uniform** class produces tensors that are filled with uniformly distributed random numbers.

The following table lists the supported parameters for the **Uniform** class.

Table 24 **Uniform** class parameters

Parameter	Required	Description	Type	Values
range	No	Range of produced random numbers.	**list of float**	The default value is [-1.000000, 1.000000,].

3.8. Utility Operators

The utility operators is a collection of common operations to help with casting, copying, and debugging.

3.8.1. Cast

The **Cast** class casts the tensor to a different type.

The following table lists the supported parameters for the **Cast** class.

Table 25 **Cast** class parameters

Parameter	Required	Description	Type	Values
dtype	Yes	Output data type.	**dali.types.DALIDataType**	

3.8.2. Copy

The **Copy** class makes a copy of the input tensor.

There are no parameters for this class.

3.8.3. DummyOp

The **DummyOp** class is the dummy operator for testing.

The following table lists the supported parameters for the **DummyOp** class.

Table 26 **DummyOp** class parameters

Parameter	Required	Description	Type	Values
num_outputs	No	Number of outputs.		The default value is2.

3.8.4. DumpImage

The **DumpImage** class saves the images in batch to disk in PPM format.

The following table lists the supported parameters for the **DumpImage** class.

Table 27 **DumpImage** class parameters

Parameter	Required	Description	Type	Values
input_layout	No	Layout of input images.	**dali.types.DALITensorLayout**	The default value is**NHWC**.
suffix	No	Suffix to be added to the output file names.	**string**	

3.8.5. ExternalSource

The **ExternalSource** class enables externally provided data to be passed as an input to the pipeline.

There are no parameters for this class.

Chapter 4. SAMPLES

The **`dali/examples`** directory contains a series of examples, in the form of Jupyter notebooks, that show different features of DALI. The examples also show how to use DALI to interface with the deep learning frameworks.

4.1. Working With Deep Learning Frameworks

DALI enables frameworks, such as MXNet, PyTorch, and TensorFlow, to bypass the native input data pipeline across deep learning tasks such as managing data, designing, and training neural networks on multi-GPU systems.

In order to minimize the steps needed to replace the native data pipeline in deep learning frameworks, DALI provides a built-in plugins to simplify integration into MXNet, PyTorch, and TensorFlow frameworks.

4.1.1. Data Loading: TFRecord

What Does This Sample Do?

The **`DataLoading-TFRecord.ipynb`** sample demonstrates how to use DALI with data stored in TensorFlow TFRecord file format.

Where Is This Sample Located?

The **`DataLoading-TFRecord.ipynb`** sample is located in the **`examples`** directory.

4.1.2. PyTorch: Execute ResNet-50 Pipeline

What Does This Sample Do?

The **`Pytorch-ResNet50.md`** sample uses the results from a DALI pipeline to train a classification model, such as ResNet-50, using PyTorch.

Where Is This Sample Located?

The **`Pytorch-ResNet50.md`** sample is located in the **`examples/pytorch/`** directory.

4.1.3. TensorFlow: Execute ResNet-50 Pipeline

What Does This Sample Do?

DALI provides a custom TensorFlow op called **`DALIIterator`**. The purpose of the **`DALIIterator`** op is to understand both DALI tensors and TensorFlow tensors and transform one into the other.

The **`TensorFlow-ResNet50.ipynb`** sample demonstrates how to use DALI with TensorFlow training. There are three parts to this sample:

- Define the DALI pipeline
- Give the pipeline to the custom op
- Use the custom op in a TensorFlow graph to train a classification model, such as ResNet-50.

Where Is This Sample Located?

The **`TensorFlow-ResNet50.ipynb`** sample is located in the **`examples/tensorflow/`** directory.

4.2. Data Loading: LMDB

What Does This Sample Do?

The **`DataLoading-LMDB.ipynb`** sample demonstrates how to use DALI with data stored in LMDB in either Caffe or Caffe2 format.

Where Is This Sample Located?

The **`DataLoading-LMDB.ipynb`** sample is located in the **`examples`** directory.

4.3. Data Loading: RecordIO

What Does This Sample Do?

The **`DataLoading-RecordIO.ipynb`** sample demonstrates how to use DALI with data stored in MXNet RecordIO file format.

Where Is This Sample Located?

The **`DataLoading-RecordIO.ipynb`** sample is located in the **`examples`** directory.

4.4. MXNet: Execute ResNet-50 Pipeline

What Does This Sample Do?

The **`MXNet-ResNet50.ipynb`** sample uses the results from a DALI pipeline to train a classification model, such as ResNet-50, using MXNet.

Where Is This Sample Located?

The **`MXNet-ResNet50.ipynb`** sample is located in the **`examples/mxnet/`** directory.

4.5. Serialization

What Does This Sample Do?

The **`Serialization.ipynb`** sample demonstrates how to serialize the pipeline defined in Python so you can use it with either C API or training with TensorFlow.

Where Is This Sample Located?

The **`Serialization.ipynb`** sample is located in the **`examples`** directory.

4.6. Augmentation Gallery

What Does This Sample Do?

The **`AugmentationGallery.ipynb`** sample lists the different image augmentations you can use in DALI.

Where Is This Sample Located?

The **`AugmentationGallery.ipynb`** sample is located in the **`examples`** directory.