# TABLE OF CONTENTS

Chapter 1. Overview Of NVCaffe................................................................. 1
   1.1. Contents Of The NVCaffe Container......................................................... 1
Chapter 2. Pulling An NVCaffe™ Container..................................................... 2
Chapter 3. Verifying NVCaffe........................................................................ 3
Chapter 4. Running An NVCaffe Container...................................................... 4
   4.1. Running An NVCaffe Container On A Cluster.......................................... 4
Chapter 5. Customizing And Extending NVCaffe.............................................. 5
   5.1. Benefits And Limitations To Customizing NVCaffe................................. 6
   5.2. Example 1: Customizing NVCaffe Using Dockerfile............................... 6
   5.3. Example 2: Customizing NVCaffe Using docker commit.......................... 7
Chapter 6. NVCaffe Parameters................................................................. 11
   6.1. Parameter Definitions............................................................................ 11
   6.2. Added and Modified Parameters......................................................... 12
Chapter 7. Troubleshooting......................................................................... 38
Chapter 1.
OVERVIEW OF NVCaffe

Caffe™ is a deep-learning framework made with flexibility, speed, and modularity in mind. It was originally developed by the Berkeley Vision and Learning Center (BVLC) and by community contributors.

NVCaffe™ is an NVIDIA-maintained fork of BVLC Caffe tuned for NVIDIA GPUs, particularly in multi-GPU configurations.

For information about the optimizations and changes that have been made to NVCaffe, see the Deep Learning Frameworks Release Notes.

1.1. Contents Of The NVCaffe Container

This image contains source and binaries for NVCaffe. The pre-built and installed version of NVCaffe is located in the /usr/local/[bin,share,lib] directories. The complete source code is located in /opt/caffe directory.

This container image also includes pycaffe, which makes the NVCaffe interfaces available for use through Python.

The NVIDIA® Collective Communications Library™ (NCCL) library and NVCaffe bindings for NCCL are installed in this container, and models using multiple GPUs will automatically leverage this library for fast parallel training.
Chapter 2.
PULLING AN NVCAFFE™ CONTAINER

To pull an NV caffe container, see Pulling A Container.
Chapter 3.
VERIFYING NVCAFFE

After you run NVCAffe, it is a good idea to verify that the container image is running correctly. To do this, issue the following commands from within the container:

```
# cd /opt/caffe
# data/mnist/get_mnist.sh
# examples/mnist/create_mnist.sh
# examples/mnist/train_lenet.sh
```

If everything is running correctly, NVCAffe should download and create a data set, and then start training LeNet. If the training is successful, you will see a code similar to the following towards the end of the output:

```
I0402 15:08:01.016016 33 solver.cpp:431] Iteration 10000, loss = 0.0342847
I0402 15:08:01.016043 33 solver.cpp:453] Iteration 10000, Testing net (#0)
I0402 15:08:01.085050 38 data_reader.cpp:128] Restarting data pre-fetching
I0402 15:08:01.087720 33 solver.cpp:543] Test net output #0: accuracy = 0.9587
I0402 15:08:01.087751 33 solver.cpp:543] Test net output #1: loss = 0.130223 (* 1 = 0.130223 loss)
I0402 15:08:01.087767 33 caffe.cpp:239] Solver performance on device 0: 498.3 * 64 = 3.189e+04 img/sec
I0402 15:08:01.087780 33 caffe.cpp:242] Optimization Done in 24s
```

If NVCAffe is not running properly, or failed during the pulling phase, check your internet connection.
Chapter 4.
RUNNING AN NVCAFFE CONTAINER

To run an NVCAFFE container, see Running NVCAFFE.

4.1. Running An NVCAFFE Container On A Cluster

NVCAFFE supports training on multiple nodes using OpenMPI version 2.0 protocol, however, you cannot specify the number of threads per process because NVCAFFE has its own thread manager (currently it runs one worker thread per GPU). For example:

```
dgx job submit --name jobname --volume <src>:<dst> --tasks 48
--clusterid <id> --gpu 8 --cpu 64 --mem 480 --image <tag> --nc "mpirun
--bind-to none -np 48 -pernode --tag-output caffe train --solver
solver.prototxt --gpu all >> /logs/caffe.log 2>&1"
```
Chapter 5.
CUSTOMIZING AND EXTENDING NVCAFFE

The nvidia-docker images come prepackaged, tuned, and ready to run; however, you may want to build a new image from scratch or augment an existing image with custom code, libraries, data, or settings for your corporate infrastructure. This section will guide you through exercises that will highlight how to create a container from scratch, customize a container, extend a deep learning framework to add features, develop some code using that extended framework from the developer environment, then package that code as a versioned release.

By default, you do not need to build a container. The NGC container registry NVIDIA container repository, nvcr.io, has a number of containers that can be used immediately including containers for deep learning as well as containers with just the CUDA® Toolkit™.

One of the great things about containers is that they can be used as starting points for creating new containers. This can be referred to as customizing or extending a container. You can create a container completely from scratch, however, since these containers are likely to run on GPUs, it is recommended that you at least start with a nvcr.io container that contains the OS and CUDA®. However, you are not limited to this and can create a container that runs on the CPUs which does not use the GPUs. In this case, you can start with a bare OS container from another location such as Docker. To make development easier, you can still start with a container with CUDA; it is just not used when the container is used.

The customized or extended containers can be saved to a user's private container repository. They can also be shared with other users but this requires some administrator help.
It is important to note that all nvidia-docker deep learning framework images include the source to build the framework itself as well as all of the prerequisites.

Attention Do not install an NVIDIA driver into the Docker image at docker build time. The nvidia-docker is essentially a wrapper around docker that transparently provisions a container with the necessary components to execute code on the GPU.

A best-practice is to avoid docker commit usage for developing new docker images, and to use Dockerfiles instead. The Dockerfile method provides visibility and capability to efficiently version-control changes made during development of a Docker image. The Docker commit method is appropriate for short-lived, disposable images only.

For more information on writing a Docker file, see the best practices documentation.

5.1. Benefits And Limitations To Customizing NVCaffe

You can customize a container to fit your specific needs for numerous reasons; for example, you depend upon specific software that is not included in the container that NVIDIA provides. No matter your reasons, you can customize a container.

The container images do not contain sample data-sets or sample model definitions unless they are included with the framework source. Be sure to check the container for sample data-sets or models.

5.2. Example 1: Customizing NVCaffe Using Dockerfile

This example uses a Dockerfile to customize the NVCaffe container in nvcr.io. Before customizing the container, you should ensure the NVCaffe 17.03 container has been loaded into the registry using the docker pull command before proceeding.

```bash
$ docker pull nvcr.io/nvidia/caffe:17.03
```

The Docker containers on nvcr.io also provide a sample Dockerfile that explains how to patch a framework and rebuild the Docker image. In the directory, `workspace/docker-examples`, there are two sample Dockerfiles that you can use. The first one, Dockerfile.addpackages, can be used to add packages to the NVCaffe image. The second one, Dockerfile.customtensorflow, illustrates how to patch NVCaffe and rebuild the image. For this example, we will use the Dockerfile.customcaffe file as a template for customizing a container.

1. Create a working directory called my_docker_images on your local hard drive.
2. Open a text editor and create a file called `Dockerfile`. Save the file to your working directory.

3. Open your `Dockerfile` again and include the following lines in the file:

```bash
FROM nvcr.io/nvidia/caffe:17.03
# APPLY CUSTOMER PATCHES TO CAFFE
# Bring in changes from outside container to /tmp
# (assumes my-caffe-modifications.patch is in same directory as Dockerfile)
#COPY my-caffe-modifications.patch /tmp

# Change working directory to NVCaffe source path
WORKDIR /opt/caffe

# Apply modifications
#RUN patch -p1 < /tmp/my-caffe-modifications.patch

# Note that the default workspace for caffe is /workspace
RUN mkdir build && cd build &&
    cmake -DCMAKE_INSTALL_PREFIX:PATH=/usr/local -DUSE_NCCL=ON
    -DUSE_CUDNN=ON -DCUDA_ARCH_NAME=Manual -DCUDA_ARCH_BIN="35 52 60 61"
    -DCUDA_ARCH_PTX="61" .. && 
    make -j"$(nproc)" install && 
    make clean && 
    cd .. && rm -rf build

# Reset default working directory
WORKDIR /workspace
```

Save the file.

4. Build the image using the `docker build` command and specify the repository name and tag. In the following example, the repository name is `corp/caffe` and the tag is `17.03.1PlusChanges`. For the case, the command would be the following:

```bash
$ docker build -t corp/caffe:17.03.1PlusChanges .
```

5. Run the Docker image using the `nvidia-docker run` command. For example:

```bash
docker run --gpus all -ti --rm corp/caffe:17.03.1PlusChanges .
```

### 5.3. Example 2: Customizing NVCaffe Using `docker commit`

This example uses the `docker commit` command to flush the current state of the container to a Docker image. This is not a recommended best practice, however, this is useful when you have a container running to which you have made changes and want to save them. In this example, we are using the `apt-get` tag to install packages which requires that the user run as root.

▶️ The NVCaffe image release 17.04 is used in the example instructions for illustrative purposes.
1. Pull the Docker container from the `nvcr.io` repository to the DGX™ system. For example, the following command will pull the NVCaffe container:

   ```bash
   $ docker pull nvcr.io/nvidia/caffe:17.04
   ```

2. Run the container on the DGX system using `nvidia-docker`.

   ```bash
   docker run --gpus all -ti nvcr.io/nvidia/caffe:17.04
   ```

3. You should now be the root user in the container (notice the prompt). You can use the `apt` command to pull down a package and put it in the container.

   ```bash
   # apt-get update
   # apt install octave
   ```

   You have to first issue `apt-get update` before you install Octave using `apt`.

4. Exit the workspace.

   ```bash
   # exit
   ```

5. Display the list of containers using `docker ps -a`. As an example, here is some of the output from the `docker ps -a` command:
6. Now you can create a new image from the container that is running where you have installed Octave. You can commit the container with the following command.

```
$ docker commit 1fe228556a97 nvcr.io/nvidia/caffe:17.04 sha256:0248470f46e22af7e6c90b65fdee6b4c6362d08779a0b8c84f45de53a6ce9294
```

7. Display the list of images.

```
$ docker images
```

```
REPOSITORY                  TAG       IMAGE ID       ...  
nvidian_sas/caffe_octave    17.04     75211f8ec225 ...
```

8. To verify, let’s run the container again and see if Octave is actually there.

```
docker run --gpus all -ti nvidian_sas/caffe_octave:17.04
```

```
======
== NVIDIA Caffe ==
==================
NVIDIA Release 17.04 (build 26740)
```

Container image Copyright (c) 2017, NVIDIA CORPORATION. All rights reserved. Copyright (c) 2014, 2015, The Regents of the University of California (Regents) All rights reserved.

Various files include modifications (c) NVIDIA CORPORATION. All rights reserved. NVIDIA modifications are covered by the license terms that apply to the underlying project or file.

NOTE: The SHMEM allocation limit is set to the default of 64MB. This may be insufficient for NVIDIA Caffe. NVIDIA recommends the use of the following flags:
```
nvidia-docker run --shm-size=1g --ulimit memlock=-1 --ulimit stack=67108864 ...
```

Since the octave prompt displayed, Octave is installed.
9. If you are using a DGX-1 or DGX Station, and you want to save the container into your private repository on `nvcr.io` (Docker uses the phrase “push”), then you can use the `docker push ...` command.

```bash
$ docker push nvcr.io/nvidian_sas/caffe_octave:17.04
```

Note that you cannot push the container to `nvcr.io` if you are using the NGC. However, you can push it to your own private repository. The new Docker image is now available for use. You can check your local Docker repository for it.
Chapter 6.
NVCAFFE PARAMETERS

Within the NVCAffe container, there is a caffe.proto file that NVIDIA has updated. The modifications that NVIDIA made are described in the following sections. These added parameters are to help implement the optimizations of the container into your environment.

6.1. Parameter Definitions

Ensure you are familiar with the following parameters.

**Boolean**
A boolean value is a data type. There are two types of boolean values; true and false. If the string argument is not null, the object types value is true. Anything other than a string type of null results in a false type.

**Enumerated**
There are two types of enumerated values:

- **Type** affects the math and storage precision. The values acceptable are:
  - DOUBLE
    - 64-bit (also referred to as double precision) floating point type.
  - FLOAT
    - 32-bit floating point type. This is the most common and default one.
  - FLOAT16
    - 16-bit floating point type.
- **Engine** affects the compute engine. The values acceptable are:
  - DEFAULT
    - Default implementation of algorithms and routines. Usually equals to CAFFE or CUDNN.
  - CAFFE
    - Basic CPU or GPU based implementation.
  - CUDNN
    - Advanced implementation based on highly optimized CUDA® Deep Neural Network library™ (cuDNN).
Floating Point Number
There is no fixed number of digits before or after the decimal point. Meaning the decimal point can float. The decimal point can be placed anywhere.

Integer
An integer is any whole number that is positive, negative, or zero.

String
A string is simply a set of characters with no relation to length.

6.2. Added and Modified Parameters

In addition to the parameters within the `caffe.proto` file included in the BVLC Caffe™ container, the following parameters have either been added for modified with the NVCaffe™ version.

For parameters not mentioned in this guide, see BVLC.

6.2.1. SolverParameter

The SolverParameter sets the solvers parameters.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>enum</td>
</tr>
<tr>
<td>Required</td>
<td>yes</td>
</tr>
<tr>
<td>Default value</td>
<td>FLOAT</td>
</tr>
<tr>
<td>Level</td>
<td>solver</td>
</tr>
</tbody>
</table>

Usage Example

```plaintext
net: "train_val_fp16.prototxt"
test_iter: 1042
test_interval: 5000
base_lr: 0.03
lr_policy: "poly"
power: 2
display: 100
max_iter: 75000
momentum: 0.9
weight_decay: 0.0005
snapshot: 150000
snapshot_prefix: "snapshots/alexnet_fp16"
solver_mode: GPU
random_seed: 1371
snapshot_after_train: false
solver_data_type: FLOAT16
```

6.2.1.1. solver_data_type

The solver_data_type parameter is the type used for storing weights and history.
### NVCaffe Parameters

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>enum</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>FLOAT</td>
</tr>
<tr>
<td>Level</td>
<td>solver</td>
</tr>
</tbody>
</table>

**Usage Example**

```plaintext
solver_data_type: FLOAT16
```

#### 6.2.1.2. min_lr

The min_lr parameter ensures that the learning rate (lr) threshold is larger than 0.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>float</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>0</td>
</tr>
<tr>
<td>Level</td>
<td>solver</td>
</tr>
</tbody>
</table>

**Usage Example**

```plaintext
net: "train_val_fp16.prototxt"
test_iter: 1042
test_interval: 5000
base_lr: 0.03
min_lr: 1e-5
lr_policy: "poly"
...
```

#### 6.2.1.3. store_blobs_in_old_format

If set to true, the store_blobs_in_old_format parameter:

1. Stores blobs in an old, less efficient BVLC-compatible format.
2. FP16 blobs are converted to FP32 and stored in the data container.
3. FP32 blobs are stored in the data container.
4. FP64 blobs are stored in the double_data container.

In rare cases, when the model is trained in NVCaffe™ but deployed to BVLC Caffe™, this parameter ensures there is BVLC compatibility.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>boolean</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
</tbody>
</table>
### 6.2.1.4. LARC - Layer-wise Adaptive Rate Control

The layer-wise adaptive rate control (LARC) is an algorithm for automatic adjustment of local learning rate per learning parameters set (weights, bias) per layer: $lr = \eta \frac{\|w\|_2}{\|\nabla w\|_2}$. After computing $lr$, it resets the update rate according to the policy defined in `larc_policy` [default = "scale"].

#### 6.2.1.4.1. larc [default = false];

The `larc` [default = false]; algorithm defines if you want LARC to be turned on or off. If set to `true`, LARC is turned on.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>boolean</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>false</td>
</tr>
<tr>
<td>Level</td>
<td>solver</td>
</tr>
</tbody>
</table>

Usage Example

```
larc: true
larc_policy: "clip"
larc_eta: 0.002
```

#### 6.2.1.4.2. larc_policy [default = "scale"];

The `larc_policy` [default = "scale"]; algorithm affects the update rate. For more information about the algorithm definition, see LARC - Layer-wise Adaptive Rate Control. Possible values are `scale` and `clip`.

- **scale**
  - The `scale` policy computes the update rate as $\lambda = lr * gr$.

- **clip**
  - The `clip` policy computes the update rate as $\lambda = \min(lr, gr)$. Here, $gr$ is the global rate.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
</tbody>
</table>
### NVCaffe Parameters

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
<th>Required</th>
<th>Default value</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Default value</td>
<td>scale</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level</td>
<td>solver</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Usage Example

```plaintext
larc: true
larc_policy: "clip"
larc_eta: 0.002
```

6.2.1.4.3. larc_eta = 51 [default = 0.001];

See section LARC - Layer-wise Adaptive Rate Control for the algorithm definition. The floating point coefficient formula is $\eta$ in the $lr = \frac{\eta}{\|w\|_2}$ ($\|\nabla w\|_2$).

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>float</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>0</td>
</tr>
<tr>
<td>Level</td>
<td>solver</td>
</tr>
</tbody>
</table>

#### Usage Example

```plaintext
larc: true
larc_policy: "clip"
larc_eta: 0.002
```

6.2.1.5. Adaptive Weight Decay

The adaptive weight decay parameters define the variable weight decay value. The fixed policy (default) keeps the same value. The polynomial policy makes the value a variable.

6.2.1.5.1. weight_decay_policy

The weight_decay_policy parameter sets the policy for the weight decay value. Possible values are fixed and poly.

**fixed**

The fixed policy keeps the value set by the weight_decay parameter.

**poly**

The poly starts from zero and ends at the value set by the weight_decay parameter using polynomial of power set by the weight_decay_power parameter.
### NVCaffe Parameters

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>fixed</td>
</tr>
<tr>
<td>Level</td>
<td>solver</td>
</tr>
</tbody>
</table>

#### Usage Example

```plaintext
weight_decay: 2e-4
weight_decay_policy: "poly"
weight_decay_power: 1.
```

#### 6.2.1.5.2. weight_decay_power

The `weight_decay_power` parameter is the power value for the `weight_decay_policy` parameter.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>float</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>0.5</td>
</tr>
<tr>
<td>Level</td>
<td>solver</td>
</tr>
</tbody>
</table>

#### Usage Example

```plaintext
weight_decay: 2e-4
weight_decay_policy: "poly"
weight_decay_power: 1.
```

#### 6.2.1.6. Adaptive Momentum

The adaptive momentum parameters defines the variable momentum value. The fixed policy (default) keeps the same value. The polynomial policy makes the value a variable.

#### 6.2.1.6.1. momentum_policy

The `momentum_policy` parameter sets the policy for the momentum value. Possible values are `fixed` and `poly`.

- **fixed**
  - The `fixed` policy keeps the value set by the momentum parameter.

- **poly**
  - The `poly` starts from the momentum parameter and ends at the value set by the `max_momentum` parameter. It uses the polynomial of power set by the `momentum_power` parameter.
### 6.2.1.6.1. momentum

The momentum parameter is the momentum update parameter. It is updated in the form of a momentum update.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>fixed</td>
</tr>
<tr>
<td>Level</td>
<td>solver</td>
</tr>
</tbody>
</table>

#### Usage Example

```plaintext
momentum: 0.9
momentum_policy: "poly"
momentum_power: 2.
max_momentum: 0.95
```

### 6.2.1.6.2. momentum_power

The momentum_power parameter is the power value of the momentum_policy parameter.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>float</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>1.0</td>
</tr>
<tr>
<td>Level</td>
<td>solver</td>
</tr>
</tbody>
</table>

#### Usage Example

```plaintext
momentum: 0.9
momentum_policy: "poly"
momentum_power: 2.
max_momentum: 0.95
```

### 6.2.1.6.3. max_momentum

The max_momentum parameter is the maximum value for momentum.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>float</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>0.99</td>
</tr>
<tr>
<td>Level</td>
<td>solver</td>
</tr>
</tbody>
</table>
6.2.2. NetParameter

The NetParameter parameter controls the layers that make up the net. If NetParameter is set, it controls all of the layers within the LayerParameter. Each of the configurations, including connectivity and behavior, is specified as a LayerParameter.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>type</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>FLOAT</td>
</tr>
<tr>
<td>Level</td>
<td>layer</td>
</tr>
</tbody>
</table>

Usage Example

```plaintext
name: "AlexNet-fp16"
default_forward_type: FLOAT16
default_backward_type: FLOAT16
default_forward_math: FLOAT
default_backward_math: FLOAT
```

6.2.2.1. default_forward_type

The default_forward_type parameter is the default data storage type used in forward pass for all layers.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>type</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>FLOAT</td>
</tr>
<tr>
<td>Level</td>
<td>net</td>
</tr>
</tbody>
</table>

Usage Example

```plaintext
default_forward_type: FLOAT16
```

6.2.2.2. default_backward_type
The default_backward_type parameter is the default data storage type used in backward pass for all layers.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>type</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>FLOAT</td>
</tr>
<tr>
<td>Level</td>
<td>net</td>
</tr>
</tbody>
</table>

Usage Example

default_backward_type: FLOAT16

6.2.2.3. default_forward_math

The default_forward_math parameter is the default data compute type used in forward pass for all layers.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>type</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>FLOAT</td>
</tr>
<tr>
<td>Level</td>
<td>net</td>
</tr>
</tbody>
</table>

Usage Example

default_forward_math: FLOAT16

6.2.2.4. default_backward_math

The default_backward_math parameter is the default data compute type used in backward pass for all layers.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>type</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>FLOAT</td>
</tr>
<tr>
<td>Level</td>
<td>net</td>
</tr>
</tbody>
</table>

Usage Example

default_backward_math: FLOAT16
6.2.2.5. reduce_buckets

The reduce_buckets parameter sets the approximate number of buckets to combine layers into. While using multiple GPUs, a reduction process is run after every iteration. For better performance, multiple layers are unified in buckets. The default value should work for the majority of nets.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>integer</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>6</td>
</tr>
<tr>
<td>Level</td>
<td>net</td>
</tr>
</tbody>
</table>

Usage Example

reduce_buckets: 10

6.2.2.6. conv_algos_override

The conv_algos_override parameter overrides the convolution algorithms to values that are specified by the user rather than ones suggested by the seeker. For example, if set to a non-negative value, it enforces using the algorithm by the index provided. It has priority over CuDNNConvolutionAlgorithmSeeker and essentially disables seeking. The index should correspond the ordinal in structures:

- cudnnConvolutionFwdAlgo_t
- cudnnConvolutionBwdDataAlgo_t
- cudnnConvolutionBwdFilterAlgo_t

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>&quot;-1,-1,-1&quot;</td>
</tr>
<tr>
<td>Level</td>
<td>layer</td>
</tr>
</tbody>
</table>

Usage Example

```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
    decay_mult: 1
  }
}  ```
```{code}
}

```{code}

```{code}
}

NVCaffe Parameters

6.2.2.7. global_grad_scale

The global_grad_scale parameter defines the constant $C$ used to improve the precision of back-propagation for float16 data storage. Gradients of loss function are multiplied by $C$ before back-propagation starts; then gradients with regards to weights are divided by $C$ accordingly before they are used for weight update.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>float</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>1</td>
</tr>
<tr>
<td>Level</td>
<td>net</td>
</tr>
</tbody>
</table>

Usage Example

```
```{code}

```

6.2.2.8. global_grad_scale_adaptive

The global_grad_scale_adaptive parameter if set to true, gradients are scaled by $C*L$ where:

- $L$ is L_2 norm of all gradients in a Net
- $C$ is the value set by the global_grad_scale parameter

This usually helps to improve accuracy of mixed precision training.
<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>boolean</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>false</td>
</tr>
<tr>
<td>Level</td>
<td>net</td>
</tr>
</tbody>
</table>

**Usage Example**

```
global_grad_scale_adaptive = true
```

### 6.2.2.9. default_cudnn_math_override

The `default_cudnn_math_override` parameter sets the default `cudnn_math_override` value for every layer if applicable. For more information, see `cudnn_math_override`.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>integer</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>-1</td>
</tr>
<tr>
<td>Level</td>
<td>net</td>
</tr>
</tbody>
</table>

**Usage Example**

```
name: "MyNet"
default_forward_type: FLOAT16
default_backward_type: FLOAT16
default_forward_math: FLOAT
default_backward_math: FLOAT
default_cudnn_math_override: 0
```

### 6.2.2.10. eltwise_mem_sharing

The `eltwise_mem_sharing` parameter is a "smart" memory sharing for EltwiseLayer which boosts performance by reducing memory consumption and copying. Majority of models (like ResNet) work with this mode. Some rare models do not, therefore, carefully test before using. Set to `true` by default in order to maintain the current behavior.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>boolean</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>true</td>
</tr>
<tr>
<td>Level</td>
<td>net</td>
</tr>
</tbody>
</table>
Usage Example

name: "MyNet"
eltwise_mem_sharing: false

6.2.3. LayerParameter

The LayerParameter parameter consists of the following memory storage types:

- forward_type
- backward_type
- forward_math
- backward_math

The internal match types works for those layers where the internal match type could be different compared to the forward or backward type. For example, pseudo fp32 mode in convolution layers.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>type</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>FLOAT</td>
</tr>
<tr>
<td>Level</td>
<td>layer</td>
</tr>
</tbody>
</table>

Usage Example

layer {
    ....
    forward_type: FLOAT
    backward_type: FLOAT
    ....
}

6.2.3.1. forward_type

The forward_type parameter is the output data storage type used by this layer in forward pass.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>type</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>FLOAT</td>
</tr>
<tr>
<td>Level</td>
<td>layer</td>
</tr>
</tbody>
</table>

Usage Example

forward_type: FLOAT16
### 6.2.3.2. backward_type

The backward_type parameter is the output data storage type used by this layer in backward pass.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>type</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>FLOAT</td>
</tr>
<tr>
<td>Level</td>
<td>layer</td>
</tr>
</tbody>
</table>

**Usage Example**

```plaintext
backward_type: FLOAT16
```

### 6.2.3.3. forward_math

The forward_math parameter computes the precision type used by this layer in forward pass.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>type</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>FLOAT</td>
</tr>
<tr>
<td>Level</td>
<td>layer</td>
</tr>
</tbody>
</table>

**Usage Example**

```plaintext
forward_math: FLOAT16
```

### 6.2.3.4. backward_math

The backward_math parameter computes the precision type used by this layer in backward pass.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>type</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>FLOAT</td>
</tr>
<tr>
<td>Level</td>
<td>layer</td>
</tr>
</tbody>
</table>
Usage Example

backward_math: FLOAT16

6.2.3.5. cudnn_math_override

The cudnn_math_override parameter sets the default cudnnMathType_t value for all CUDA® Deep Neural Network library™ (cuDNN)-based computations in the current layer, if applicable, otherwise, it is ignored. If negative or omitted, it assumes implicit default and allows optimizers like cudnnFindConvolution*AlgorithmEx to choose the best type. If set to zero, it enforces using CUDNN_DEFAULT_MATH everywhere in the current layer. If set to one, it enforces using CUDNN_TENSOR_OP_MATH everywhere in the current layer.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>integer</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>-1</td>
</tr>
<tr>
<td>Level</td>
<td>layer</td>
</tr>
</tbody>
</table>

Usage Example

layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  convolution_param {
    num_output: 32
    kernel_size: 3
    stride: 2
    weight_filler {
      type: "xavier"
    }
    bias_term: false
  }
  cudnn_math_override: 1
}

6.2.4. TransformationParameter

The TransformationParameter parameter consists of settings that can be used for data pre-processing. It stores parameters that are used to apply transformation to the data layers data.

Usage Example

transform_param {
  mirror: true
  crop_size: 227
}
6.2.4.1. use_gpu_transform

The use_gpu_transform parameter runs the transform, synchronously, on the GPU.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>boolean</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>false</td>
</tr>
<tr>
<td>Level</td>
<td>layer &gt; transform_param</td>
</tr>
</tbody>
</table>

Usage Example

use_gpu_transform: true

6.2.4.2. img_rand_resize_lower

The img_rand_resize_lower parameter specifies that the variable-sized input image should be randomly resized. The aspect ratio of the resized image is preserved, but the shortest side of the resized image is uniformly sampled from the closed interval between img_rand_resize_lower and img_rand_resize_upper.

This parameter is currently incompatible with mean_file.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>integer</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>0</td>
</tr>
<tr>
<td>Level</td>
<td>layer &gt; transform_param</td>
</tr>
</tbody>
</table>

Usage Example

img_rand_resize_lower: 256

6.2.4.3. img_rand_resize_upper

The img_rand_resize_upper parameter specifies that the variable-sized input image should be randomly resized. The aspect ratio of the resized image is preserved, but the
shortest side of the resized image is uniformly sampled from the closed interval between img_rand_resize_lower and img_rand_resize_upper.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>integer</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>0</td>
</tr>
<tr>
<td>Level</td>
<td>layer &gt; transform_param</td>
</tr>
</tbody>
</table>

Usage Example

img_rand_resize_upper: 480

6.2.4.4. rand_resize_ratio_lower

The rand_resize_ratio_lower parameter sets lower limit for randomly generated ratio $R$ so that the length of the longer side is set to the length of the shorter side, multiplied by $R$. If applied to a square, the shorter side is chosen randomly. The $\{1,1\}$ pair of limits means resize the image to a square (by shortest side). Values less than 1 are ignored.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>float</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>0</td>
</tr>
<tr>
<td>Level</td>
<td>layer &gt; transform_param</td>
</tr>
</tbody>
</table>

Usage Example

rand_resize_ratio_lower: 1

6.2.4.5. rand_resize_ratio_upper

The rand_resize_ratio_upper parameter sets the upper limit for randomly generated ratio $R$ so that the length of the longer side is set to the length of the shorter side, multiplied by $R$. If applied to a square, the shorter side is chosen randomly. The $\{1,1\}$ pair of limits means resize the image to a square (by shortest side). Values less than 1 are ignored.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>float</td>
</tr>
</tbody>
</table>
**Rand Resize Ratio Upper**

```
rand_resize_ratio_upper: 1.2
```

**6.2.4.6. Vertical Stretch Lower**

The vertical_stretch_lower parameter limits for randomly generated vertical stretch. In other words, \( \text{height} \times \text{vertical_stretch} \) where \( \text{vertical_stretch} = \text{Rand}(\text{vertical_stretch_lower}) \). Pair \( {1,1} \) means no action is needed.

```
Setting    Value
Type       float
Required   no
Default value 1
Level     layer > transform_param
```

**Usage Example**

```
vertical_stretch_lower: 0.8
```

**6.2.4.7. Vertical Stretch Upper**

The vertical_stretch_upper parameter limits for randomly generated vertical stretch. In other words, \( \text{height} \times \text{vertical_stretch} \) where \( \text{vertical_stretch} = \text{Rand}(\text{vertical_stretch_upper}) \). Pair \( {1,1} \) means no action is needed.

```
Setting    Value
Type       float
Required   no
Default value 1
Level     layer > transform_param
```

**Usage Example**

```
vertical_stretch_upper: 1.2
```

**6.2.4.8. Horizontal Stretch Lower**
The horizontal_stretch_lower parameter limits for randomly generated horizontal stretch. In other words, \( \text{width} \times= \text{horizontal_stretch} \) where \( \text{horizontal_stretch} = \text{Rand(} \text{horizontal_stretch\_lower} \text{)} \). Pair \( \{1,1\} \) means no action is needed.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>float</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>1</td>
</tr>
<tr>
<td>Level</td>
<td>( \text{layer} &gt; \text{transform_param} )</td>
</tr>
</tbody>
</table>

**Usage Example**

`horizontal_stretch_lower: 0.8`

### 6.2.4.9. horizontal_stretch_upper

The horizontal_stretch_upper parameter limits for randomly generated horizontal stretch. In other words, \( \text{width} \times= \text{horizontal_stretch} \) where \( \text{horizontal_stretch} = \text{Rand(} \text{horizontal_stretch\_upper} \text{)} \). Pair \( \{1,1\} \) means no action is needed.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>float</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>1</td>
</tr>
<tr>
<td>Level</td>
<td>( \text{layer} &gt; \text{transform_param} )</td>
</tr>
</tbody>
</table>

**Usage Example**

`horizontal_stretch_upper: 1.2`

### 6.2.4.10. interpolation_algo_down

The interpolation_algo_down parameter sets the image resizing algorithm used by OpenCV to downscale an image.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>enum InterpolationAlgo { INTER_NEAREST INTER_LINEAR INTER_CUBIC INTER_AREA }</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>INTER_NEAREST</td>
</tr>
</tbody>
</table>

www.nvidia.com

NVCaffe
6.2.4.11. interpolation_algo_up

The interpolation_algo_up parameter sets the image resizing algorithm used by OpenCV to upscale an image.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>enum InterpolationAlgo { INTER_NEAREST, INTER_LINEAR, INTER_CUBIC, INTER_AREA }</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>INTER_CUBIC</td>
</tr>
<tr>
<td>Level</td>
<td>layer &gt; transform_param</td>
</tr>
</tbody>
</table>

Usage Example

interpolation_algo_up: INTER_LINEAR

6.2.4.12. allow_upscale

The allow_upscale parameter enables you to upscale images.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>boolean</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>false</td>
</tr>
<tr>
<td>Level</td>
<td>layer &gt; transform_param</td>
</tr>
</tbody>
</table>

Usage Example

allow_upscale : true

6.2.5. BatchNormParameter

In NVCAFFE version 0.15, it was required to explicitly set \( lr\_mul: 0 \) and \( decay\_mul\_v: 0 \) for certain BatchNormParameter parameters (global_mean and global variance) to prevent their modification by gradient solvers. In version 0.16, this is done automatically, therefore, these parameters are not needed any more.
In NVCaffe version 0.15, it was also required that `bottom` and `top` contain different values. Although it is recommended that they remain different, this requirement is now optional.

**Usage Example**

```java
layer {
    name: "conv1_bn"
    type: "BatchNorm"
    bottom: "conv1"
    top: "conv1_bn"
    batch_norm_param {
        moving_average_fraction: 0.9
        eps: 0.0001
        scale_bias: true
    }
}
```

6.2.5.1. `scale_bias`

The `scale_bias` parameter allows you to fuse batch normalization and scale layers. Beginning in version 0.16, batch normalization supports both NVCaffe and BVLC Caffe.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>boolean</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>false</td>
</tr>
<tr>
<td>Level</td>
<td>layer</td>
</tr>
</tbody>
</table>

**Usage Example**

```
layer {
    name: "bn"
    type: "BatchNorm"
    bottom: "conv"
    top: "bn"
    batch_norm_param {
        moving_average_fraction: 0.9
        eps: 0.0001
        scale_bias: true
    }
}
```

6.2.6. ConvolutionParameter

The ConvolutionParameter parameter specifies which cuDNN routine should be used to find the best convolution algorithm.
### NVCaffe Parameters

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>CuDNNConvolutionAlgorithmSeeker</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>FINDEX</td>
</tr>
<tr>
<td>Level</td>
<td>LayerParameter</td>
</tr>
</tbody>
</table>

#### Usage Example

```plaintext
convolution_param {
    num_output: 96
    kernel_size: 11
    stride: 4
    weight_filler {
        type: "gaussian"
        std: 0.01
    }
    bias_filler {
        type: "constant"
        value: 0
    }
    cudnn_convolution_algo_seeker: FINDEX
}
```

### 6.2.6.1. cudnn_convolution_algo_seeker

The cudnn_convolution_algo_seeker parameter specifies which cuDNN routine should be used to find the best convolution algorithm.

The most common use case scenario for NVCaffe is the image recognition. The convolution layer is the layer that stores the algorithms to process the images. The algorithm seeker has two engines:

- **GET**
  - GET is the heuristic engine.

- **FINDEX**
  - FINDEX makes real calls and real assessments and takes a few seconds to assess all possible algorithms for each and every convolutional layer.
Usage Example

layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 96
    kernel_size: 11
    stride: 4
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
    cudnn_convolution_algo_seeker: FINDEX
  }
}

6.2.7. DataParameter

The DataParameter belongs to the Data Layer’s LayerParameter settings. Besides regular BVLC settings, it contains the following performance related settings, threads and parser_threads.

Usage Example

data_param {
  source: "/raid/caffe_imagenet_lmdb/ilsvrc12_train_lmdb"
  batch_size: 1024
  backend: LMDB
}

6.2.7.1. threads

The threads parameter is the number of Data Transformer threads per GPU executed by DataLayer. Prior to 17.04, the default is 3, which is the optimal value for the majority of nets.

Data Transformer is a component converting source data. It is compute intensive, therefore, if you think that DataLayer under-performs, set the value to 4.

In 17.04, the default is 0. If set to 0, NVCaffe optimizes it automatically.
### 6.2.7.2. parser_threads

The `parser_threads` parameter is the number of Data Reader and Parser threads per GPU. Prior to 17.04, the default is 2, which is the optimal value for the majority of nets.

Asynchronous Data Reader is an NVcaffe component. It dramatically increases read speed. Google Protocol Buffers parser is a component that de-serializes raw data that is read by the Reader into a structure called Datum. If you observe messages like `Waiting for Datum`, increase the setting value to 4 or higher.

In 17.04, the default is 0. If set to 0, NVcaffe optimizes it automatically.

#### Usage Example

```
parser_threads: 4
```

### 6.2.7.3. cache

The `cache` parameter ensures that the data is read once and put into the host memory. If the data does not fit in the host memory, the cache data is dropped and the NVcaffe model reads the data from the database.

#### Usage Example

```
parser_threads: 4
```

```
6.2.7.2. parser_threads

The parser_threads parameter is the number of Data Reader and Parser threads per GPU. Prior to 17.04, the default is 2, which is the optimal value for the majority of nets.

Asynchronous Data Reader is an NVcaffe component. It dramatically increases read speed. Google Protocol Buffers parser is a component that de-serializes raw data that is read by the Reader into a structure called Datum. If you observe messages like Waiting for Datum, increase the setting value to 4 or higher.

In 17.04, the default is 0. If set to 0, NVcaffe optimizes it automatically.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>unsigned integer</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>0</td>
</tr>
<tr>
<td>Level</td>
<td>DataParameter of DataLayer</td>
</tr>
</tbody>
</table>

#### Usage Example

```
parser_threads: 4
```

```
6.2.7.3. cache

The cache parameter ensures that the data is read once and put into the host memory. If the data does not fit in the host memory, the cache data is dropped and the NVcaffe model reads the data from the database.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>boolean</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>false</td>
</tr>
<tr>
<td>Level</td>
<td>DataParameter of DataLayer</td>
</tr>
</tbody>
</table>
Usage Example

    cache: true

6.2.7.4. shuffle

The shuffle parameter is ignored if the cache parameter is set to false. Shuffling is a data augmentation technique that improves accuracy of training your network. If cache does not fit in the host memory, shuffling will be cancelled.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>boolean</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>false</td>
</tr>
<tr>
<td>Level</td>
<td>DataParameter of data layer</td>
</tr>
</tbody>
</table>

Usage Example

    shuffle: true

6.2.8. ImageDataParameter

The ImageDataParameter belongs to the ImageDataLayer's LayerParameter settings. Besides regular BVLC settings, it contains the following performance related settings, threads and cache.

Usage Example

    image_data_param {
      source: "train-jpeg_map.txt"
      batch_size: 128
      shuffle: true
      new_height: 227
      new_width: 227
      cache: true
    }

6.2.8.1. threads

The threads parameter is the number of Data Transformer threads per GPU executed by the ImageDataLayer. The default is 4, which is the optimal value for the majority of nets. Data Transformer is a component converting source data. It is compute intensive, therefore, if you think that ImageDataLayer underperforms, set it to larger value.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>unsigned integer</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
</tbody>
</table>
## NVCaffe Parameters

### 6.2.8.2. cache

The cache parameter ensures that the data is read once and put into the host memory. If the data does not fit in the host memory, the program stops.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>boolean</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>false</td>
</tr>
</tbody>
</table>

#### Usage Example

```plaintext
cache: true
```

### 6.2.9. ELUParameter

The ELUParameter stores parameters used by ELULayer.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>structure</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>1.</td>
</tr>
<tr>
<td>Level</td>
<td>layer</td>
</tr>
</tbody>
</table>

#### Usage Example

```plaintext
Layer{
    name: "selu"
    type: "ELU"
    bottom: "bottom"
    top: "top"
    elu_param {
        alpha: 1.6733
    }
}
```
6.2.9.1. lambda

The lambda parameter is used for Scaled Exponential Linear Unit (SELU). SELU is a non-linear activation layer, which is defined as follows:

- If input $x \geq 0$ then output
- If input $x < 0$ then output

$$selu(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha e^x - \alpha & \text{if } x \leq 0 \end{cases}$$

**Figure 1** Scaled Exponential Linear Unit (SELU)

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>float</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Default value</td>
<td>1.</td>
</tr>
<tr>
<td>Level</td>
<td>layer</td>
</tr>
</tbody>
</table>

**Usage Example**

```plaintext
Layer{
    name: "selu"
    type: "ELU"
    bottom: "bottom"
    top: "top"
    elu_param {
        alpha: 1.6733
        lambda: 1.0507
    }
}
```
Chapter 7.
TROUBLESHOOTING

For more information about NVcaffe, including tutorials, documentation, and examples, see the Caffe website.

NVcaffe typically utilizes the same input formats and configuration parameters as Caffe, therefore, community-authored materials and pre-trained models for Caffe usually can be applied to NVcaffe as well.

For the latest NVcaffe Release Notes, see the Deep Learning Documentation website.
Notice

THE INFORMATION IN THIS GUIDE AND ALL OTHER INFORMATION CONTAINED IN NVIDIA DOCUMENTATION REFERENCED IN THIS GUIDE IS PROVIDED “AS IS.” NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE INFORMATION FOR THE PRODUCT, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the product described in this guide shall be limited in accordance with the NVIDIA terms and conditions of sale for the product.

THE NVIDIA PRODUCT DESCRIBED IN THIS GUIDE IS NOT FAULT TOLERANT AND IS NOT DESIGNED, MANUFACTURED OR INTENDED FOR USE IN CONNECTION WITH THE DESIGN, CONSTRUCTION, MAINTENANCE, AND/OR OPERATION OF ANY SYSTEM WHERE THE USE OR A FAILURE OF SUCH SYSTEM COULD RESULT IN A SITUATION THAT THREATENS THE SAFETY OF HUMAN LIFE OR SEVERE PHYSICAL HARM OR PROPERTY DAMAGE (INCLUDING, FOR EXAMPLE, USE IN CONNECTION WITH ANY NUCLEAR, AVIONICS, LIFE SUPPORT OR OTHER LIFE CRITICAL APPLICATION). NVIDIA EXPRESSLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS FOR SUCH HIGH RISK USES. NVIDIA SHALL NOT BE LIABLE TO CUSTOMER OR ANY THIRD PARTY, IN WHOLE OR IN PART, FOR ANY CLAIMS OR DAMAGES ARISING FROM SUCH HIGH RISK USES.

NVIDIA makes no representation or warranty that the product described in this guide will be suitable for any specified use without further testing or modification. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to ensure the product is suitable and fit for the application planned by customer and to do the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer’s product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this guide. NVIDIA does not accept any liability related to any default, damage, costs or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this guide, or (ii) customer product designs.

Other than the right for customer to use the information in this guide with the product, no other license, either expressed or implied, is hereby granted by NVIDIA under this guide. Reproduction of information in this guide is permissible only if reproduction is approved by NVIDIA in writing, is reproduced without alteration, and is accompanied by all associated conditions, limitations, and notices.

Trademarks

NVIDIA, the NVIDIA logo, and cuBLAS, CUDA, cuDNN, DALI, DIGITS, DGX, DGX-1, DGX-2, DGX Station, DLPprof, Jetson, Kepler, Maxwell, NCCL, Nsight Compute, Nsight Systems, NvCaffe, PerfWorks, Pascal, SDK Manager, Tegra, TensorRT, TensorRT Inference Server, Tesla, TF-TRT, and Volta are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2020 NVIDIA Corporation. All rights reserved.

www.nvidia.com