Caffe

Generated by Doxygen 1.8.13

Contents

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

Namespace Documentation

4.1 boost Namespace Reference

4.1.1 Detailed Description

Forward declare boost::thread instead of including boost/thread.hpp to avoid a boost/NVCC issues (#1009, #1010) on OSX.

4.2 caffe Namespace Reference

A layer factory that allows one to register layers. During runtime, registered layers can be called by passing a LayerParameter protobuffer to the CreateLayer function:

Namespaces

- SolverAction

  Enumeration of actions that a client of the Solver may request by implementing the Solver's action request function, which a client may optionally provide in order to request early termination or saving a snapshot without exiting. In the executable caffe, this mechanism is used to allow the snapshot to be saved when stopping execution with a SIGINT (Ctrl-C).

Classes

- class AbsValLayer

  Computes $y = |x|$.
- class AccuracyLayer

  Computes the classification accuracy for a one-of-many classification task.
- class AdaDeltaSolver
- class AdaGradSolver
- class AdamSolver

  AdamSolver, an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. Described in [1].
- class ArgMaxLayer

Compute the index of the K max values for each datum across all dimensions $(C \times H \times W)$.
- class BaseConvolutionLayer

  Abstract base class that factors out the BLAS code common to ConvolutionLayer and DeconvolutionLayer.
- class BaseDataLayer

  Provides base for data layers that feed blobs to the Net.
- class BasePrefetchingDataLayer
- class Batch
- class BatchNormLayer

  Normalizes the input to have 0-mean and/or unit (1) variance across the batch.
- class BatchReindexLayer

  Index into the input blob along its first axis.
- class BiasLayer

  Computes a sum of two input Blobs, with the shape of the latter Blob "broadcast" to match the shape of the former. Equivalent to tiling the latter Blob, then computing the elementwise sum.
- class BilinearFiller

  Fills a Blob with coefficients for bilinear interpolation.
- class Blob

  A wrapper around SyncedMemory holders serving as the basic computational unit through which Layers, Nets, and Solvers interact.
- class BlockingQueue
- class BNLLLayer

  Computes $y = x + \log(1 + \exp(-x))$ if $x > 0$; $y = \log(1 + \exp(x))$ otherwise.
- class Caffe
- class ClipLayer

  Clip: $y = \max(min, \min(max, x))$.
- class ConcatLayer

  Takes at least two Blobs and concatenates them along either the num or channel dimension, outputting the result.
- class ConstantFiller

  Fills a Blob with constant values $x = 0$.
- class ContrastiveLossLayer

  Computes the contrastive loss $E = \frac{1}{2N} \sum\limits_{n=1}^{N} \left(y\right) d^2 + \left(1-y\right) \max \left(margin-d, 0\right)^2$ where $d = \left| \left| a_n - b_n \right| \right|_2$. This can be used to train siamese networks.
- class ConvolutionLayer

  Convolves the input image with a bank of learned filters, and (optionally) adds biases.
- class CPUTimer
- class CropLayer

  Takes a Blob and crop it, to the shape specified by the second input Blob, across all dimensions after the specified axis.
- class DataLayer
- class DataTransformer

  Applies common transformations to the input data, such as scaling, mirroring, substracting the image mean...
- class DeconvolutionLayer

  Convolve the input with a bank of learned filters, and (optionally) add biases, treating filters and convolution parameters in the opposite sense as ConvolutionLayer.
- class DropoutLayer

  During training only, sets a random portion of x to 0, adjusting the rest of the vector magnitude accordingly.
- class DummyDataLayer

  Provides data to the Net generated by a Filler.
- class EltwiseLayer

  Compute elementwise operations, such as product and sum, along multiple input Blobs.
- class ELULayer

Exponential Linear Unit non-linearity $y = \left\{ \begin{array}{lr} x & \mathrm{if} \; x > 0 \\ \alpha (\exp(x)-1) & \mathrm{if} \; x \le 0 \end{array} \right.$.

- class EmbedLayer

  A layer for learning "embeddings" of one-hot vector input. Equivalent to an InnerProductLayer with one-hot vectors as input, but for efficiency the input is the "hot" index of each column itself.

- class EuclideanLossLayer

  Computes the Euclidean (L2) loss $E = \frac{1}{2N} \sum\limits_{n=1}^N \left| \left| \hat{y}_n - y_n \right| \right|_2^2$ *for real-valued regression tasks.*

- class ExpLayer

  Computes $y = \gamma ^ {\alpha x + \beta}$*, as specified by the scale* α*, shift* β*, and base* γ*.*

- class Filler

  Fills a Blob with constant or randomly-generated data.

- class FilterLayer

  Takes two+ Blobs, interprets last Blob as a selector and filter remaining Blobs accordingly with selector data (0 means that the corresponding item has to be filtered, non-zero means that corresponding item needs to stay).

- class FlattenLayer

  Reshapes the input Blob into flat vectors.

- class GaussianFiller

  Fills a Blob with Gaussian-distributed values $x = a$*.*

- class HDF5DataLayer

  Provides data to the Net from HDF5 files.

- class HDF5OutputLayer

  Write blobs to disk as HDF5 files.

- class HingeLossLayer

  Computes the hinge loss for a one-of-many classification task.

- class Im2colLayer

  A helper for image operations that rearranges image regions into column vectors. Used by ConvolutionLayer to perform convolution by matrix multiplication.

- class ImageDataLayer

  Provides data to the Net from image files.

- class InfogainLossLayer

  A generalization of SoftmaxWithLossLayer that takes an "information gain" (infogain) matrix specifying the "value" of all label pairs.

- class InnerProductLayer

  Also known as a "fully-connected" layer, computes an inner product with a set of learned weights, and (optionally) adds biases.

- class InputLayer

  Provides data to the Net by assigning tops directly.

- class InternalThread
- class Layer

  An interface for the units of computation which can be composed into a Net.

- class LayerRegisterer
- class LayerRegistry
- class LogLayer

  Computes $y = log_{\gamma}(\alpha x + \beta)$*, as specified by the scale* α*, shift* β*, and base* γ*.*

- class LossLayer

  An interface for Layers that take two Blobs as input – usually (1) predictions and (2) ground-truth labels – and output a singleton Blob representing the loss.

- class LRNLayer

  Normalize the input in a local region across or within feature maps.

- class LSTMLayer

  Processes sequential inputs using a "Long Short-Term Memory" (LSTM) [1] style recurrent neural network (RNN). Implemented by unrolling the LSTM computation through time.

- class LSTMUnitLayer

  A helper for LSTMLayer: computes a single timestep of the non-linearity of the LSTM, producing the updated cell and hidden states.
- class MemoryDataLayer

  Provides data to the Net from memory.
- class MSRAFiller

  Fills a Blob with values $x \sim N(0, \sigma^2)$ where σ^2 is set inversely proportional to number of incoming nodes, outgoing nodes, or their average.
- class MultinomialLogisticLossLayer

  Computes the multinomial logistic loss for a one-of-many classification task, directly taking a predicted probability distribution as input.
- class MVNLayer

  Normalizes the input to have 0-mean and/or unit (1) variance.
- class NesterovSolver
- class Net

  Connects Layers together into a directed acyclic graph (DAG) specified by a NetParameter.
- class NeuronLayer

  An interface for layers that take one blob as input (x) and produce one equally-sized blob as output (y), where each element of the output depends only on the corresponding input element.
- class ParameterLayer
- class PoolingLayer

  Pools the input image by taking the max, average, etc. within regions.
- class PositiveUnitballFiller

  Fills a Blob with values $x \in [0, 1]$ such that $\forall i \sum_j x_{ij} = 1$.
- class PowerLayer

  Computes $y = (\alpha x + \beta)^\gamma$, as specified by the scale α, shift β, and power γ.
- class PReLULayer

  Parameterized Rectified Linear Unit non-linearity $y_i = \max(0, x_i) + a_i \min(0, x_i)$. The differences from ReLULayer are 1) negative slopes are learnable though backprop and 2) negative slopes can vary across channels. The number of axes of input blob should be greater than or equal to 2. The 1st axis (0-based) is seen as channels.
- class PythonLayer
- class RecurrentLayer

  An abstract class for implementing recurrent behavior inside of an unrolled network. This Layer type cannot be instantiated – instead, you should use one of its implementations which defines the recurrent architecture, such as RNNLayer or LSTMLayer.
- class ReductionLayer

  Compute "reductions" – operations that return a scalar output Blob for an input Blob of arbitrary size, such as the sum, absolute sum, and sum of squares.
- class ReLULayer

  Rectified Linear Unit non-linearity $y = \max(0, x)$. The simple max is fast to compute, and the function does not saturate.
- class ReshapeLayer
- class RMSPropSolver
- class RNNLayer

  Processes time-varying inputs using a simple recurrent neural network (RNN). Implemented as a network unrolling the RNN computation in time.
- class ScaleLayer

  Computes the elementwise product of two input Blobs, with the shape of the latter Blob "broadcast" to match the shape of the former. Equivalent to tiling the latter Blob, then computing the elementwise product. Note: for efficiency and convenience, this layer can additionally perform a "broadcast" sum too when `bias_term: true` is set.
- class SGDSolver

  Optimizes the parameters of a Net using stochastic gradient descent (SGD) with momentum.
- class SigmoidCrossEntropyLossLayer

Computes the cross-entropy (logistic) loss $E = \frac{-1}{n} \sum\limits_{n=1}^{N} \left[p_n \log \hat{p}_n + (1 - p_n) \log(1 - \hat{p}_n) \right]$*, often used for predicting targets interpreted as probabilities.*

- class SigmoidLayer

  Sigmoid function non-linearity $y = (1 + \exp(-x))^{-1}$*, a classic choice in neural networks.*

- class SignalHandler
- class SilenceLayer

  Ignores bottom blobs while producing no top blobs. (This is useful to suppress outputs during testing.)

- class SliceLayer

  Takes a Blob and slices it along either the num or channel dimension, outputting multiple sliced Blob results.

- class SoftmaxLayer

  Computes the softmax function.

- class SoftmaxWithLossLayer

  Computes the multinomial logistic loss for a one-of-many classification task, passing real-valued predictions through a softmax to get a probability distribution over classes.

- class Solver

  An interface for classes that perform optimization on Nets.

- class SolverRegisterer
- class SolverRegistry
- class SplitLayer

  Creates a "split" path in the network by copying the bottom Blob into multiple top Blobs to be used by multiple consuming layers.

- class SPPLayer

  Does spatial pyramid pooling on the input image by taking the max, average, etc. within regions so that the result vector of different sized images are of the same size.

- class SwishLayer

  Swish non-linearity $y = x\sigma(\beta x)$*. A novel activation function that tends to work better than ReLU [1].*

- class SyncedMemory

  Manages memory allocation and synchronization between the host (CPU) and device (GPU).

- class TanHLayer

  TanH hyperbolic tangent non-linearity $y = \frac{\exp(2x)-1}{\exp(2x)+1}$*, popular in auto-encoders.*

- class ThresholdLayer

  Tests whether the input exceeds a threshold: outputs 1 for inputs above threshold; 0 otherwise.

- class TileLayer

  Copy a Blob along specified dimensions.

- class Timer
- class UniformFiller

  Fills a Blob with uniformly distributed values $x \sim U(a, b)$*.*

- class WindowDataLayer

  Provides data to the Net from windows of images files, specified by a window data file. This layer is DEPRECATED and only kept for archival purposes for use by the original R-CNN.

- class XavierFiller

  Fills a Blob with values $x \sim U(-a, +a)$ *where* a *is set inversely proportional to number of incoming nodes, outgoing nodes, or their average.*

Typedefs

- typedef boost::function< SolverAction::Enum()> ActionCallback

  Type of a function that returns a Solver Action enumeration.

- typedef boost::mt19937 **rng_t**

Functions

- void **GlobalInit** (int ∗pargc, char ∗∗∗pargv)
- template< typename Dtype >
  Filler< Dtype > ∗ GetFiller (const FillerParameter &param)

  Get a specific filler from the specification given in FillerParameter.
- void **CaffeMallocHost** (void ∗∗ptr, size_t size, bool ∗use_cuda)
- void **CaffeFreeHost** (void ∗ptr, bool use_cuda)
- const char ∗ **cublasGetErrorString** (cublasStatus_t error)
- const char ∗ **curandGetErrorString** (curandStatus_t error)
- int **CAFFE_GET_BLOCKS** (const int N)
- std::string **format_int** (int n, int numberOfLeadingZeros=0)
- template< typename Dtype >
  void **im2col_nd_cpu** (const Dtype ∗data_im, const int num_spatial_axes, const int ∗im_shape, const int ∗col_shape, const int ∗kernel_shape, const int ∗pad, const int ∗stride, const int ∗dilation, Dtype ∗data_col)
- template< typename Dtype >
  void **im2col_cpu** (const Dtype ∗data_im, const int channels, const int height, const int width, const int kernel_h, const int kernel_w, const int pad_h, const int pad_w, const int stride_h, const int stride_w, const int dilation_h, const int dilation_w, Dtype ∗data_col)
- template< typename Dtype >
  void **col2im_nd_cpu** (const Dtype ∗data_col, const int num_spatial_axes, const int ∗im_shape, const int ∗col_shape, const int ∗kernel_shape, const int ∗pad, const int ∗stride, const int ∗dilation, Dtype ∗data_im)
- template< typename Dtype >
  void **col2im_cpu** (const Dtype ∗data_col, const int channels, const int height, const int width, const int kernel_h, const int kernel_w, const int pad_h, const int pad_w, const int stride_h, const int stride_w, const int dilation_h, const int dilation_w, Dtype ∗data_im)
- template< typename Dtype >
  void **im2col_nd_gpu** (const Dtype ∗data_im, const int num_spatial_axes, const int col_size, const int ∗im↩_shape, const int ∗col_shape, const int ∗kernel_shape, const int ∗pad, const int ∗stride, const int ∗dilation, Dtype ∗data_col)
- template< typename Dtype >
  void **im2col_gpu** (const Dtype ∗data_im, const int channels, const int height, const int width, const int kernel_h, const int kernel_w, const int pad_h, const int pad_w, const int stride_h, const int stride_w, const int dilation_h, const int dilation_w, Dtype ∗data_col)
- template< typename Dtype >
  void **col2im_nd_gpu** (const Dtype ∗data_col, const int num_spatial_axes, const int im_size, const int ∗im↩_shape, const int ∗col_shape, const int ∗kernel_shape, const int ∗pad, const int ∗stride, const int ∗dilation, Dtype ∗data_im)
- template< typename Dtype >
  void **col2im_gpu** (const Dtype ∗data_col, const int channels, const int height, const int width, const int kernel_h, const int kernel_w, const int pad_h, const int pad_w, const int stride_h, const int stride_w, const int dilation_h, const int dilation_w, Dtype ∗data_im)
- void **InsertSplits** (const NetParameter &param, NetParameter ∗param_split)
- void **ConfigureSplitLayer** (const string &layer_name, const string &blob_name, const int blob_idx, const int split_count, const float loss_weight, LayerParameter ∗split_layer_param)
- string **SplitLayerName** (const string &layer_name, const string &blob_name, const int blob_idx)
- string **SplitBlobName** (const string &layer_name, const string &blob_name, const int blob_idx, const int split_idx)
- void **MakeTempDir** (string ∗temp_dirname)
- void **MakeTempFilename** (string ∗temp_filename)
- bool **ReadProtoFromTextFile** (const char ∗filename, Message ∗proto)
- bool **ReadProtoFromTextFile** (const string &filename, Message ∗proto)
- void **ReadProtoFromTextFileOrDie** (const char ∗filename, Message ∗proto)
- void **ReadProtoFromTextFileOrDie** (const string &filename, Message ∗proto)
- void **WriteProtoToTextFile** (const Message &proto, const char ∗filename)
- void **WriteProtoToTextFile** (const Message &proto, const string &filename)

- bool **ReadProtoFromBinaryFile** (const char ∗filename, Message ∗proto)
- bool **ReadProtoFromBinaryFile** (const string &filename, Message ∗proto)
- void **ReadProtoFromBinaryFileOrDie** (const char ∗filename, Message ∗proto)
- void **ReadProtoFromBinaryFileOrDie** (const string &filename, Message ∗proto)
- void **WriteProtoToBinaryFile** (const Message &proto, const char ∗filename)
- void **WriteProtoToBinaryFile** (const Message &proto, const string &filename)
- bool **ReadFileToDatum** (const string &filename, const int label, Datum ∗datum)
- bool **ReadFileToDatum** (const string &filename, Datum ∗datum)
- bool **ReadImageToDatum** (const string &filename, const int label, const int height, const int width, const bool is_color, const std::string &encoding, Datum ∗datum)
- bool **ReadImageToDatum** (const string &filename, const int label, const int height, const int width, const bool is_color, Datum ∗datum)
- bool **ReadImageToDatum** (const string &filename, const int label, const int height, const int width, Datum ∗datum)
- bool **ReadImageToDatum** (const string &filename, const int label, const bool is_color, Datum ∗datum)
- bool **ReadImageToDatum** (const string &filename, const int label, Datum ∗datum)
- bool **ReadImageToDatum** (const string &filename, const int label, const std::string &encoding, Datum ∗datum)
- bool **DecodeDatumNative** (Datum ∗datum)
- bool **DecodeDatum** (Datum ∗datum, bool is_color)
- template<typename Dtype >
  void **caffe_cpu_gemm** (const CBLAS_TRANSPOSE TransA, const CBLAS_TRANSPOSE TransB, const int M, const int N, const int K, const Dtype alpha, const Dtype ∗A, const Dtype ∗B, const Dtype beta, Dtype ∗C)
- template<typename Dtype >
  void **caffe_cpu_gemv** (const CBLAS_TRANSPOSE TransA, const int M, const int N, const Dtype alpha, const Dtype ∗A, const Dtype ∗x, const Dtype beta, Dtype ∗y)
- template<typename Dtype >
  void **caffe_axpy** (const int N, const Dtype alpha, const Dtype ∗X, Dtype ∗Y)
- template<typename Dtype >
  void **caffe_cpu_axpby** (const int N, const Dtype alpha, const Dtype ∗X, const Dtype beta, Dtype ∗Y)
- template<typename Dtype >
  void **caffe_copy** (const int N, const Dtype ∗X, Dtype ∗Y)
- template<typename Dtype >
  void **caffe_set** (const int N, const Dtype alpha, Dtype ∗X)
- void **caffe_memset** (const size_t N, const int alpha, void ∗X)
- template<typename Dtype >
  void **caffe_add_scalar** (const int N, const Dtype alpha, Dtype ∗X)
- template<typename Dtype >
  void **caffe_scal** (const int N, const Dtype alpha, Dtype ∗X)
- template<typename Dtype >
  void **caffe_sqr** (const int N, const Dtype ∗a, Dtype ∗y)
- template<typename Dtype >
  void **caffe_sqrt** (const int N, const Dtype ∗a, Dtype ∗y)
- template<typename Dtype >
  void **caffe_add** (const int N, const Dtype ∗a, const Dtype ∗b, Dtype ∗y)
- template<typename Dtype >
  void **caffe_sub** (const int N, const Dtype ∗a, const Dtype ∗b, Dtype ∗y)
- template<typename Dtype >
  void **caffe_mul** (const int N, const Dtype ∗a, const Dtype ∗b, Dtype ∗y)
- template<typename Dtype >
  void **caffe_div** (const int N, const Dtype ∗a, const Dtype ∗b, Dtype ∗y)
- template<typename Dtype >
  void **caffe_powx** (const int n, const Dtype ∗a, const Dtype b, Dtype ∗y)
- unsigned int **caffe_rng_rand** ()
- template<typename Dtype >
  Dtype **caffe_nextafter** (const Dtype b)

- template<typename Dtype >
  void **caffe_rng_uniform** (const int n, const Dtype a, const Dtype b, Dtype ∗r)
- template<typename Dtype >
  void **caffe_rng_gaussian** (const int n, const Dtype mu, const Dtype sigma, Dtype ∗r)
- template<typename Dtype >
  void **caffe_rng_bernoulli** (const int n, const Dtype p, int ∗r)
- template<typename Dtype >
  void **caffe_rng_bernoulli** (const int n, const Dtype p, unsigned int ∗r)
- template<typename Dtype >
  void **caffe_exp** (const int n, const Dtype ∗a, Dtype ∗y)
- template<typename Dtype >
  void **caffe_log** (const int n, const Dtype ∗a, Dtype ∗y)
- template<typename Dtype >
  void **caffe_abs** (const int n, const Dtype ∗a, Dtype ∗y)
- template<typename Dtype >
  Dtype **caffe_cpu_dot** (const int n, const Dtype ∗x, const Dtype ∗y)
- template<typename Dtype >
  Dtype **caffe_cpu_strided_dot** (const int n, const Dtype ∗x, const int incx, const Dtype ∗y, const int incy)
- template<typename Dtype >
  Dtype **caffe_cpu_asum** (const int n, const Dtype ∗x)
- template<typename Dtype >
  int8_t **caffe_sign** (Dtype val)
- **DEFINE_CAFFE_CPU_UNARY_FUNC** (sgnbit, y[i]=static_cast< bool >((std::signbit)(x[i]))) template< typename Dtype > void caffe_cpu_scale(const int n
- template<typename Dtype >
  void **caffe_gpu_gemm** (const CBLAS_TRANSPOSE TransA, const CBLAS_TRANSPOSE TransB, const int M, const int N, const int K, const Dtype alpha, const Dtype ∗A, const Dtype ∗B, const Dtype beta, Dtype ∗C)
- template<typename Dtype >
  void **caffe_gpu_gemv** (const CBLAS_TRANSPOSE TransA, const int M, const int N, const Dtype alpha, const Dtype ∗A, const Dtype ∗x, const Dtype beta, Dtype ∗y)
- template<typename Dtype >
  void **caffe_gpu_axpy** (const int N, const Dtype alpha, const Dtype ∗X, Dtype ∗Y)
- template<typename Dtype >
  void **caffe_gpu_axpby** (const int N, const Dtype alpha, const Dtype ∗X, const Dtype beta, Dtype ∗Y)
- void **caffe_gpu_memcpy** (const size_t N, const void ∗X, void ∗Y)
- template<typename Dtype >
  void **caffe_gpu_set** (const int N, const Dtype alpha, Dtype ∗X)
- void **caffe_gpu_memset** (const size_t N, const int alpha, void ∗X)
- template<typename Dtype >
  void **caffe_gpu_add_scalar** (const int N, const Dtype alpha, Dtype ∗X)
- template<typename Dtype >
  void **caffe_gpu_scal** (const int N, const Dtype alpha, Dtype ∗X)
- template<typename Dtype >
  void **caffe_gpu_scal** (const int N, const Dtype alpha, Dtype ∗X, cudaStream_t str)
- template<typename Dtype >
  void **caffe_gpu_add** (const int N, const Dtype ∗a, const Dtype ∗b, Dtype ∗y)
- template<typename Dtype >
  void **caffe_gpu_sub** (const int N, const Dtype ∗a, const Dtype ∗b, Dtype ∗y)
- template<typename Dtype >
  void **caffe_gpu_mul** (const int N, const Dtype ∗a, const Dtype ∗b, Dtype ∗y)
- template<typename Dtype >
  void **caffe_gpu_div** (const int N, const Dtype ∗a, const Dtype ∗b, Dtype ∗y)
- template<typename Dtype >
  void **caffe_gpu_abs** (const int n, const Dtype ∗a, Dtype ∗y)
- template<typename Dtype >
  void **caffe_gpu_exp** (const int n, const Dtype ∗a, Dtype ∗y)

- template<typename Dtype >
  void **caffe_gpu_log** (const int n, const Dtype ∗a, Dtype ∗y)
- template<typename Dtype >
  void **caffe_gpu_powx** (const int n, const Dtype ∗a, const Dtype b, Dtype ∗y)
- template<typename Dtype >
  void **caffe_gpu_sqrt** (const int n, const Dtype ∗a, Dtype ∗y)
- void **caffe_gpu_rng_uniform** (const int n, unsigned int ∗r)
- template<typename Dtype >
  void **caffe_gpu_rng_uniform** (const int n, const Dtype a, const Dtype b, Dtype ∗r)
- template<typename Dtype >
  void **caffe_gpu_rng_gaussian** (const int n, const Dtype mu, const Dtype sigma, Dtype ∗r)
- template<typename Dtype >
  void **caffe_gpu_rng_bernoulli** (const int n, const Dtype p, int ∗r)
- template<typename Dtype >
  void **caffe_gpu_dot** (const int n, const Dtype ∗x, const Dtype ∗y, Dtype ∗out)
- template<typename Dtype >
  void **caffe_gpu_asum** (const int n, const Dtype ∗x, Dtype ∗y)
- template<typename Dtype >
  void **caffe_gpu_sign** (const int n, const Dtype ∗x, Dtype ∗y)
- template<typename Dtype >
  void **caffe_gpu_sgnbit** (const int n, const Dtype ∗x, Dtype ∗y)
- template<typename Dtype >
  void **caffe_gpu_fabs** (const int n, const Dtype ∗x, Dtype ∗y)
- template<typename Dtype >
  void **caffe_gpu_scale** (const int n, const Dtype alpha, const Dtype ∗x, Dtype ∗y)
- rng_t ∗ **caffe_rng** ()
- template<class RandomAccessIterator , class RandomGenerator >
  void **shuffle** (RandomAccessIterator begin, RandomAccessIterator end, RandomGenerator ∗gen)
- template<class RandomAccessIterator >
  void **shuffle** (RandomAccessIterator begin, RandomAccessIterator end)
- bool **NetNeedsUpgrade** (const NetParameter &net_param)
- bool **UpgradeNetAsNeeded** (const string &param_file, NetParameter ∗param)
- void **ReadNetParamsFromTextFileOrDie** (const string &param_file, NetParameter ∗param)
- void **ReadNetParamsFromBinaryFileOrDie** (const string &param_file, NetParameter ∗param)
- bool **NetNeedsV0ToV1Upgrade** (const NetParameter &net_param)
- bool **UpgradeV0Net** (const NetParameter &v0_net_param, NetParameter ∗net_param)
- void **UpgradeV0PaddingLayers** (const NetParameter &param, NetParameter ∗param_upgraded_pad)
- bool **UpgradeV0LayerParameter** (const V1LayerParameter &v0_layer_connection, V1LayerParameter ∗layer_param)
- V1LayerParameter_LayerType **UpgradeV0LayerType** (const string &type)
- bool **NetNeedsDataUpgrade** (const NetParameter &net_param)
- void **UpgradeNetDataTransformation** (NetParameter ∗net_param)
- bool **NetNeedsV1ToV2Upgrade** (const NetParameter &net_param)
- bool **UpgradeV1Net** (const NetParameter &v1_net_param, NetParameter ∗net_param)
- bool **UpgradeV1LayerParameter** (const V1LayerParameter &v1_layer_param, LayerParameter ∗layer_↩
  param)
- const char ∗ **UpgradeV1LayerType** (const V1LayerParameter_LayerType type)
- bool **NetNeedsInputUpgrade** (const NetParameter &net_param)
- void **UpgradeNetInput** (NetParameter ∗net_param)
- bool **NetNeedsBatchNormUpgrade** (const NetParameter &net_param)
- void **UpgradeNetBatchNorm** (NetParameter ∗net_param)
- bool **SolverNeedsTypeUpgrade** (const SolverParameter &solver_param)
- bool **UpgradeSolverType** (SolverParameter ∗solver_param)
- bool **UpgradeSolverAsNeeded** (const string &param_file, SolverParameter ∗param)
- void **ReadSolverParamsFromTextFileOrDie** (const string &param_file, SolverParameter ∗param)
- **INSTANTIATE_CLASS** (Blob)

- int64_t **cluster_seedgen** (void)
- **INSTANTIATE_CLASS** (DataTransformer)
- **INSTANTIATE_CLASS** (Layer)
- template<typename Dtype >
  shared_ptr< Layer< Dtype > > **GetConvolutionLayer** (const LayerParameter &param)
- **REGISTER_LAYER_CREATOR** (Convolution, GetConvolutionLayer)
- template<typename Dtype >
  shared_ptr< Layer< Dtype > > **GetDeconvolutionLayer** (const LayerParameter &param)
- **REGISTER_LAYER_CREATOR** (Deconvolution, GetDeconvolutionLayer)
- template<typename Dtype >
  shared_ptr< Layer< Dtype > > **GetPoolingLayer** (const LayerParameter &param)
- **REGISTER_LAYER_CREATOR** (Pooling, GetPoolingLayer)
- template<typename Dtype >
  shared_ptr< Layer< Dtype > > **GetLRNLayer** (const LayerParameter &param)
- **REGISTER_LAYER_CREATOR** (LRN, GetLRNLayer)
- template<typename Dtype >
  shared_ptr< Layer< Dtype > > **GetReLULayer** (const LayerParameter &param)
- **REGISTER_LAYER_CREATOR** (ReLU, GetReLULayer)
- template<typename Dtype >
  shared_ptr< Layer< Dtype > > **GetSigmoidLayer** (const LayerParameter &param)
- **REGISTER_LAYER_CREATOR** (Sigmoid, GetSigmoidLayer)
- template<typename Dtype >
  shared_ptr< Layer< Dtype > > **GetSoftmaxLayer** (const LayerParameter &param)
- **REGISTER_LAYER_CREATOR** (Softmax, GetSoftmaxLayer)
- template<typename Dtype >
  shared_ptr< Layer< Dtype > > **GetTanHLayer** (const LayerParameter &param)
- **REGISTER_LAYER_CREATOR** (TanH, GetTanHLayer)
- **INSTANTIATE_CLASS** (AbsValLayer)
- **REGISTER_LAYER_CLASS** (AbsVal)
- **INSTANTIATE_CLASS** (AccuracyLayer)
- **REGISTER_LAYER_CLASS** (Accuracy)
- **INSTANTIATE_CLASS** (ArgMaxLayer)
- **REGISTER_LAYER_CLASS** (ArgMax)
- **INSTANTIATE_CLASS** (BaseConvolutionLayer)
- **INSTANTIATE_CLASS** (BaseDataLayer)
- **INSTANTIATE_CLASS** (BasePrefetchingDataLayer)
- **INSTANTIATE_CLASS** (BatchNormLayer)
- **REGISTER_LAYER_CLASS** (BatchNorm)
- **INSTANTIATE_CLASS** (BatchReindexLayer)
- **REGISTER_LAYER_CLASS** (BatchReindex)
- **INSTANTIATE_CLASS** (BiasLayer)
- **REGISTER_LAYER_CLASS** (Bias)
- **INSTANTIATE_CLASS** (BNLLLayer)
- **REGISTER_LAYER_CLASS** (BNLL)
- **INSTANTIATE_CLASS** (ClipLayer)
- **REGISTER_LAYER_CLASS** (Clip)
- **INSTANTIATE_CLASS** (ConcatLayer)
- **REGISTER_LAYER_CLASS** (Concat)
- **INSTANTIATE_CLASS** (ContrastiveLossLayer)
- **REGISTER_LAYER_CLASS** (ContrastiveLoss)
- **INSTANTIATE_CLASS** (ConvolutionLayer)
- **INSTANTIATE_CLASS** (CropLayer)
- **REGISTER_LAYER_CLASS** (Crop)
- **INSTANTIATE_CLASS** (DataLayer)
- **REGISTER_LAYER_CLASS** (Data)

- **INSTANTIATE_CLASS** (DeconvolutionLayer)
- **INSTANTIATE_CLASS** (DropoutLayer)
- **REGISTER_LAYER_CLASS** (Dropout)
- **INSTANTIATE_CLASS** (DummyDataLayer)
- **REGISTER_LAYER_CLASS** (DummyData)
- **INSTANTIATE_CLASS** (EltwiseLayer)
- **REGISTER_LAYER_CLASS** (Eltwise)
- **INSTANTIATE_CLASS** (ELULayer)
- **REGISTER_LAYER_CLASS** (ELU)
- **INSTANTIATE_CLASS** (EmbedLayer)
- **REGISTER_LAYER_CLASS** (Embed)
- **INSTANTIATE_CLASS** (EuclideanLossLayer)
- **REGISTER_LAYER_CLASS** (EuclideanLoss)
- **INSTANTIATE_CLASS** (ExpLayer)
- **REGISTER_LAYER_CLASS** (Exp)
- **INSTANTIATE_CLASS** (FilterLayer)
- **REGISTER_LAYER_CLASS** (Filter)
- **INSTANTIATE_CLASS** (FlattenLayer)
- **REGISTER_LAYER_CLASS** (Flatten)
- **INSTANTIATE_CLASS** (HingeLossLayer)
- **REGISTER_LAYER_CLASS** (HingeLoss)
- **INSTANTIATE_CLASS** (Im2colLayer)
- **REGISTER_LAYER_CLASS** (Im2col)
- **INSTANTIATE_CLASS** (InfogainLossLayer)
- **REGISTER_LAYER_CLASS** (InfogainLoss)
- **INSTANTIATE_CLASS** (InnerProductLayer)
- **REGISTER_LAYER_CLASS** (InnerProduct)
- **INSTANTIATE_CLASS** (InputLayer)
- **REGISTER_LAYER_CLASS** (Input)
- **INSTANTIATE_CLASS** (LogLayer)
- **REGISTER_LAYER_CLASS** (Log)
- **INSTANTIATE_CLASS** (LossLayer)
- **INSTANTIATE_CLASS** (LRNLayer)
- **INSTANTIATE_CLASS** (LSTMLayer)
- **REGISTER_LAYER_CLASS** (LSTM)
- template<typename Dtype >
  Dtype **sigmoid** (Dtype x)
- template<typename Dtype >
  Dtype **tanh** (Dtype x)
- **INSTANTIATE_CLASS** (LSTMUnitLayer)
- **REGISTER_LAYER_CLASS** (LSTMUnit)
- **INSTANTIATE_CLASS** (MemoryDataLayer)
- **REGISTER_LAYER_CLASS** (MemoryData)
- **INSTANTIATE_CLASS** (MultinomialLogisticLossLayer)
- **REGISTER_LAYER_CLASS** (MultinomialLogisticLoss)
- **INSTANTIATE_CLASS** (MVNLayer)
- **REGISTER_LAYER_CLASS** (MVN)
- **INSTANTIATE_CLASS** (NeuronLayer)
- **INSTANTIATE_CLASS** (ParameterLayer)
- **REGISTER_LAYER_CLASS** (Parameter)
- **INSTANTIATE_CLASS** (PoolingLayer)
- **INSTANTIATE_CLASS** (PowerLayer)
- **REGISTER_LAYER_CLASS** (Power)
- **INSTANTIATE_CLASS** (PReLULayer)
- **REGISTER_LAYER_CLASS** (PReLU)

- **INSTANTIATE_CLASS** (RecurrentLayer)
- **INSTANTIATE_CLASS** (ReductionLayer)
- **REGISTER_LAYER_CLASS** (Reduction)
- **INSTANTIATE_CLASS** (ReLULayer)
- **INSTANTIATE_CLASS** (ReshapeLayer)
- **REGISTER_LAYER_CLASS** (Reshape)
- **INSTANTIATE_CLASS** (RNNLayer)
- **REGISTER_LAYER_CLASS** (RNN)
- **INSTANTIATE_CLASS** (ScaleLayer)
- **REGISTER_LAYER_CLASS** (Scale)
- **INSTANTIATE_CLASS** (SigmoidCrossEntropyLossLayer)
- **REGISTER_LAYER_CLASS** (SigmoidCrossEntropyLoss)
- **INSTANTIATE_CLASS** (SigmoidLayer)
- **INSTANTIATE_CLASS** (SilenceLayer)
- **REGISTER_LAYER_CLASS** (Silence)
- **INSTANTIATE_CLASS** (SliceLayer)
- **REGISTER_LAYER_CLASS** (Slice)
- **INSTANTIATE_CLASS** (SoftmaxLayer)
- **INSTANTIATE_CLASS** (SoftmaxWithLossLayer)
- **REGISTER_LAYER_CLASS** (SoftmaxWithLoss)
- **INSTANTIATE_CLASS** (SplitLayer)
- **REGISTER_LAYER_CLASS** (Split)
- **INSTANTIATE_CLASS** (SPPLayer)
- **REGISTER_LAYER_CLASS** (SPP)
- **INSTANTIATE_CLASS** (SwishLayer)
- **REGISTER_LAYER_CLASS** (Swish)
- **INSTANTIATE_CLASS** (TanHLayer)
- **INSTANTIATE_CLASS** (ThresholdLayer)
- **REGISTER_LAYER_CLASS** (Threshold)
- **INSTANTIATE_CLASS** (TileLayer)
- **REGISTER_LAYER_CLASS** (Tile)
- **INSTANTIATE_CLASS** (Net)
- template< typename Dtype >
  void **LoadNetWeights** (shared_ptr< Net< Dtype > > net, const std::string &model_list)
- **INSTANTIATE_CLASS** (Solver)
- template< typename Dtype >
  void **adadelta_update_gpu** (int N, Dtype *g, Dtype *h, Dtype *h2, Dtype momentum, Dtype delta, Dtype local_rate)
- **INSTANTIATE_CLASS** (AdaDeltaSolver)
- **REGISTER_SOLVER_CLASS** (AdaDelta)
- template< typename Dtype >
  void **adagrad_update_gpu** (int N, Dtype *g, Dtype *h, Dtype delta, Dtype local_rate)
- **INSTANTIATE_CLASS** (AdaGradSolver)
- **REGISTER_SOLVER_CLASS** (AdaGrad)
- template< typename Dtype >
  void **adam_update_gpu** (int N, Dtype *g, Dtype *m, Dtype *v, Dtype beta1, Dtype beta2, Dtype eps_hat, Dtype corrected_local_rate)
- **INSTANTIATE_CLASS** (AdamSolver)
- **REGISTER_SOLVER_CLASS** (Adam)
- template< typename Dtype >
  void **nesterov_update_gpu** (int N, Dtype *g, Dtype *h, Dtype momentum, Dtype local_rate)
- **INSTANTIATE_CLASS** (NesterovSolver)
- **REGISTER_SOLVER_CLASS** (Nesterov)
- template< typename Dtype >
  void **rmsprop_update_gpu** (int N, Dtype *g, Dtype *h, Dtype rms_decay, Dtype delta, Dtype local_rate)

- **INSTANTIATE_CLASS** (RMSPropSolver)
- **REGISTER_SOLVER_CLASS** (RMSProp)
- template<typename Dtype >
  void **sgd_update_gpu** (int N, Dtype ∗g, Dtype ∗h, Dtype momentum, Dtype local_rate)
- **INSTANTIATE_CLASS** (SGDSolver)
- **REGISTER_SOLVER_CLASS** (SGD)
- bool **is_a_ge_zero_and_a_lt_b** (int a, int b)
- template void **im2col_cpu**< **float** > (const float ∗data_im, const int channels, const int height, const int width, const int kernel_h, const int kernel_w, const int pad_h, const int pad_w, const int stride_h, const int stride_w, const int dilation_h, const int dilation_w, float ∗data_col)
- template void **im2col_cpu**< **double** > (const double ∗data_im, const int channels, const int height, const int width, const int kernel_h, const int kernel_w, const int pad_h, const int pad_w, const int stride_h, const int stride_w, const int dilation_h, const int dilation_w, double ∗data_col)
- template<typename Dtype >
  void **im2col_nd_core_cpu** (const Dtype ∗data_input, const bool im2col, const int num_spatial_axes, const int ∗im_shape, const int ∗col_shape, const int ∗kernel_shape, const int ∗pad, const int ∗stride, const int ∗dilation, Dtype ∗data_output)
- template void **im2col_nd_cpu**< **float** > (const float ∗data_im, const int num_spatial_axes, const int ∗im←_shape, const int ∗col_shape, const int ∗kernel_shape, const int ∗pad, const int ∗stride, const int ∗dilation, float ∗data_col)
- template void **im2col_nd_cpu**< **double** > (const double ∗data_im, const int num_spatial_axes, const int ∗im_shape, const int ∗col_shape, const int ∗kernel_shape, const int ∗pad, const int ∗stride, const int ∗dilation, double ∗data_col)
- template void **col2im_cpu**< **float** > (const float ∗data_col, const int channels, const int height, const int width, const int kernel_h, const int kernel_w, const int pad_h, const int pad_w, const int stride_h, const int stride_w, const int dilation_h, const int dilation_w, float ∗data_im)
- template void **col2im_cpu**< **double** > (const double ∗data_col, const int channels, const int height, const int width, const int kernel_h, const int kernel_w, const int pad_h, const int pad_w, const int stride_h, const int stride_w, const int dilation_h, const int dilation_w, double ∗data_im)
- template void **col2im_nd_cpu**< **float** > (const float ∗data_col, const int num_spatial_axes, const int ∗im←_shape, const int ∗col_shape, const int ∗kernel_shape, const int ∗pad, const int ∗stride, const int ∗dilation, float ∗data_im)
- template void **col2im_nd_cpu**< **double** > (const double ∗data_col, const int num_spatial_axes, const int ∗im_shape, const int ∗col_shape, const int ∗kernel_shape, const int ∗pad, const int ∗stride, const int ∗dilation, double ∗data_im)
- template<>
  void **caffe_cpu_gemm**< **float** > (const CBLAS_TRANSPOSE TransA, const CBLAS_TRANSPOSE TransB, const int M, const int N, const int K, const float alpha, const float ∗A, const float ∗B, const float beta, float ∗C)
- template<>
  void **caffe_cpu_gemm**< **double** > (const CBLAS_TRANSPOSE TransA, const CBLAS_TRANSPOSE TransB, const int M, const int N, const int K, const double alpha, const double ∗A, const double ∗B, const double beta, double ∗C)
- template<>
  void **caffe_cpu_gemv**< **float** > (const CBLAS_TRANSPOSE TransA, const int M, const int N, const float alpha, const float ∗A, const float ∗x, const float beta, float ∗y)
- template<>
  void **caffe_cpu_gemv**< **double** > (const CBLAS_TRANSPOSE TransA, const int M, const int N, const double alpha, const double ∗A, const double ∗x, const double beta, double ∗y)
- template<>
  void **caffe_axpy**< **float** > (const int N, const float alpha, const float ∗X, float ∗Y)
- template<>
  void **caffe_axpy**< **double** > (const int N, const double alpha, const double ∗X, double ∗Y)
- template void **caffe_set**< **int** > (const int N, const int alpha, int ∗Y)
- template void **caffe_set**< **float** > (const int N, const float alpha, float ∗Y)
- template void **caffe_set**< **double** > (const int N, const double alpha, double ∗Y)
- template<>
  void **caffe_add_scalar** (const int N, const float alpha, float ∗Y)

- template<>
  void **caffe_add_scalar** (const int N, const double alpha, double ∗Y)
- template void **caffe_copy**< **int** > (const int N, const int ∗X, int ∗Y)
- template void **caffe_copy**< **unsigned int** > (const int N, const unsigned int ∗X, unsigned int ∗Y)
- template void **caffe_copy**< **float** > (const int N, const float ∗X, float ∗Y)
- template void **caffe_copy**< **double** > (const int N, const double ∗X, double ∗Y)
- template<>
  void **caffe_scal**< **float** > (const int N, const float alpha, float ∗X)
- template<>
  void **caffe_scal**< **double** > (const int N, const double alpha, double ∗X)
- template<>
  void **caffe_cpu_axpby**< **float** > (const int N, const float alpha, const float ∗X, const float beta, float ∗Y)
- template<>
  void **caffe_cpu_axpby**< **double** > (const int N, const double alpha, const double ∗X, const double beta, double ∗Y)
- template<>
  void **caffe_add**< **float** > (const int n, const float ∗a, const float ∗b, float ∗y)
- template<>
  void **caffe_add**< **double** > (const int n, const double ∗a, const double ∗b, double ∗y)
- template<>
  void **caffe_sub**< **float** > (const int n, const float ∗a, const float ∗b, float ∗y)
- template<>
  void **caffe_sub**< **double** > (const int n, const double ∗a, const double ∗b, double ∗y)
- template<>
  void **caffe_mul**< **float** > (const int n, const float ∗a, const float ∗b, float ∗y)
- template<>
  void **caffe_mul**< **double** > (const int n, const double ∗a, const double ∗b, double ∗y)
- template<>
  void **caffe_div**< **float** > (const int n, const float ∗a, const float ∗b, float ∗y)
- template<>
  void **caffe_div**< **double** > (const int n, const double ∗a, const double ∗b, double ∗y)
- template<>
  void **caffe_powx**< **float** > (const int n, const float ∗a, const float b, float ∗y)
- template<>
  void **caffe_powx**< **double** > (const int n, const double ∗a, const double b, double ∗y)
- template<>
  void **caffe_sqr**< **float** > (const int n, const float ∗a, float ∗y)
- template<>
  void **caffe_sqr**< **double** > (const int n, const double ∗a, double ∗y)
- template<>
  void **caffe_sqrt**< **float** > (const int n, const float ∗a, float ∗y)
- template<>
  void **caffe_sqrt**< **double** > (const int n, const double ∗a, double ∗y)
- template<>
  void **caffe_exp**< **float** > (const int n, const float ∗a, float ∗y)
- template<>
  void **caffe_exp**< **double** > (const int n, const double ∗a, double ∗y)
- template<>
  void **caffe_log**< **float** > (const int n, const float ∗a, float ∗y)
- template<>
  void **caffe_log**< **double** > (const int n, const double ∗a, double ∗y)
- template<>
  void **caffe_abs**< **float** > (const int n, const float ∗a, float ∗y)
- template<>
  void **caffe_abs**< **double** > (const int n, const double ∗a, double ∗y)
- template float **caffe_nextafter** (const float b)

- template double **caffe_nextafter** (const double b)
- template void **caffe_rng_uniform**< **float** > (const int n, const float a, const float b, float ∗r)
- template void **caffe_rng_uniform**< **double** > (const int n, const double a, const double b, double ∗r)
- template void **caffe_rng_gaussian**< **float** > (const int n, const float mu, const float sigma, float ∗r)
- template void **caffe_rng_gaussian**< **double** > (const int n, const double mu, const double sigma, double ∗r)
- template void **caffe_rng_bernoulli**< **double** > (const int n, const double p, int ∗r)
- template void **caffe_rng_bernoulli**< **float** > (const int n, const float p, int ∗r)
- template void **caffe_rng_bernoulli**< **double** > (const int n, const double p, unsigned int ∗r)
- template void **caffe_rng_bernoulli**< **float** > (const int n, const float p, unsigned int ∗r)
- template<>
  float **caffe_cpu_strided_dot**< **float** > (const int n, const float ∗x, const int incx, const float ∗y, const int incy)
- template<>
  double **caffe_cpu_strided_dot**< **double** > (const int n, const double ∗x, const int incx, const double ∗y, const int incy)
- template float **caffe_cpu_dot**< **float** > (const int n, const float ∗x, const float ∗y)
- template double **caffe_cpu_dot**< **double** > (const int n, const double ∗x, const double ∗y)
- template<>
  float **caffe_cpu_asum**< **float** > (const int n, const float ∗x)
- template<>
  double **caffe_cpu_asum**< **double** > (const int n, const double ∗x)
- template<>
  void **caffe_cpu_scale**< **float** > (const int n, const float alpha, const float ∗x, float ∗y)
- template<>
  void **caffe_cpu_scale**< **double** > (const int n, const double alpha, const double ∗x, double ∗y)
- void **UpgradeSnapshotPrefixProperty** (const string &param_file, SolverParameter ∗param)

Variables

- const float **kLOG_THRESHOLD** = 1e-20
- const int **CAFFE_CUDA_NUM_THREADS** = 512
- const Dtype **alpha**
- const Dtype const Dtype ∗ **x**
- const Dtype const Dtype Dtype ∗ **y**
- const float **kBNLL_THRESHOLD** = 50.

4.2.1 Detailed Description

A layer factory that allows one to register layers. During runtime, registered layers can be called by passing a LayerParameter protobuffer to the CreateLayer function:

A solver factory that allows one to register solvers, similar to layer factory. During runtime, registered solvers could be called by passing a SolverParameter protobuffer to the CreateSolver function:

LayerRegistry<Dtype>::CreateLayer(param);

There are two ways to register a layer. Assuming that we have a layer like:

template <typename dtype>="“> class MyAwesomeLayer : public Layer<Dtype> { // your implementations };

and its type is its C++ class name, but without the "Layer" at the end ("MyAwesomeLayer" -> "MyAwesome").

If the layer is going to be created simply by its constructor, in your c++ file, add the following line:

REGISTER_LAYER_CLASS(MyAwesome);

Or, if the layer is going to be created by another creator function, in the format of:

template <typename dtype>=""> Layer<Dtype*> GetMyAwesomeLayer(const LayerParameter& param) { // your implementation }

(for example, when your layer has multiple backends, see GetConvolutionLayer for a use case), then you can register the creator function instead, like

REGISTER_LAYER_CREATOR(MyAwesome, GetMyAwesomeLayer)

Note that each layer type should only be registered once.

SolverRegistry<Dtype>::CreateSolver(param);

There are two ways to register a solver. Assuming that we have a solver like:

template <typename dtype>=""> class MyAwesomeSolver : public Solver<Dtype> { // your implementations };

and its type is its C++ class name, but without the "Solver" at the end ("MyAwesomeSolver" -> "MyAwesome").

If the solver is going to be created simply by its constructor, in your C++ file, add the following line:

REGISTER_SOLVER_CLASS(MyAwesome);

Or, if the solver is going to be created by another creator function, in the format of:

template <typename dtype>=""> Solver<Dtype*> GetMyAwesomeSolver(const SolverParameter& param) { // your implementation }

then you can register the creator function instead, like

REGISTER_SOLVER_CREATOR(MyAwesome, GetMyAwesomeSolver)

Note that each solver type should only be registered once.

4.2.2 Function Documentation

4.2.2.1 GetFiller()

```
template<typename Dtype >
Filler<Dtype>* caffe::GetFiller (
            const FillerParameter & param )
```

Get a specific filler from the specification given in FillerParameter.

Ideally this would be replaced by a factory pattern, but we will leave it this way for now.

4.3 caffe::SolverAction Namespace Reference

Enumeration of actions that a client of the Solver may request by implementing the Solver's action request function, which a client may optionally provide in order to request early termination or saving a snapshot without exiting. In the executable caffe, this mechanism is used to allow the snapshot to be saved when stopping execution with a SIGINT (Ctrl-C).

Enumerations

- enum **Enum** { **NONE** = 0, **STOP** = 1, **SNAPSHOT** = 2 }

4.3.1 Detailed Description

Enumeration of actions that a client of the Solver may request by implementing the Solver's action request function, which a client may optionally provide in order to request early termination or saving a snapshot without exiting. In the executable caffe, this mechanism is used to allow the snapshot to be saved when stopping execution with a SIGINT (Ctrl-C).

Chapter 5

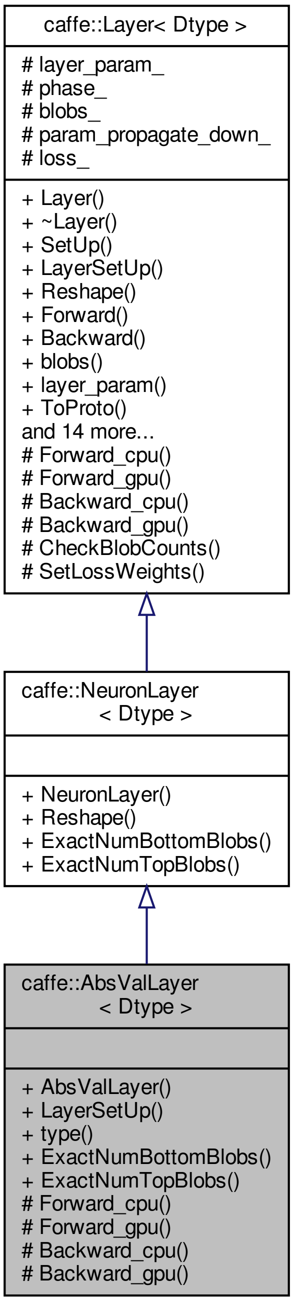
Class Documentation

5.1 caffe::AbsValLayer$<$ Dtype $>$ Class Template Reference

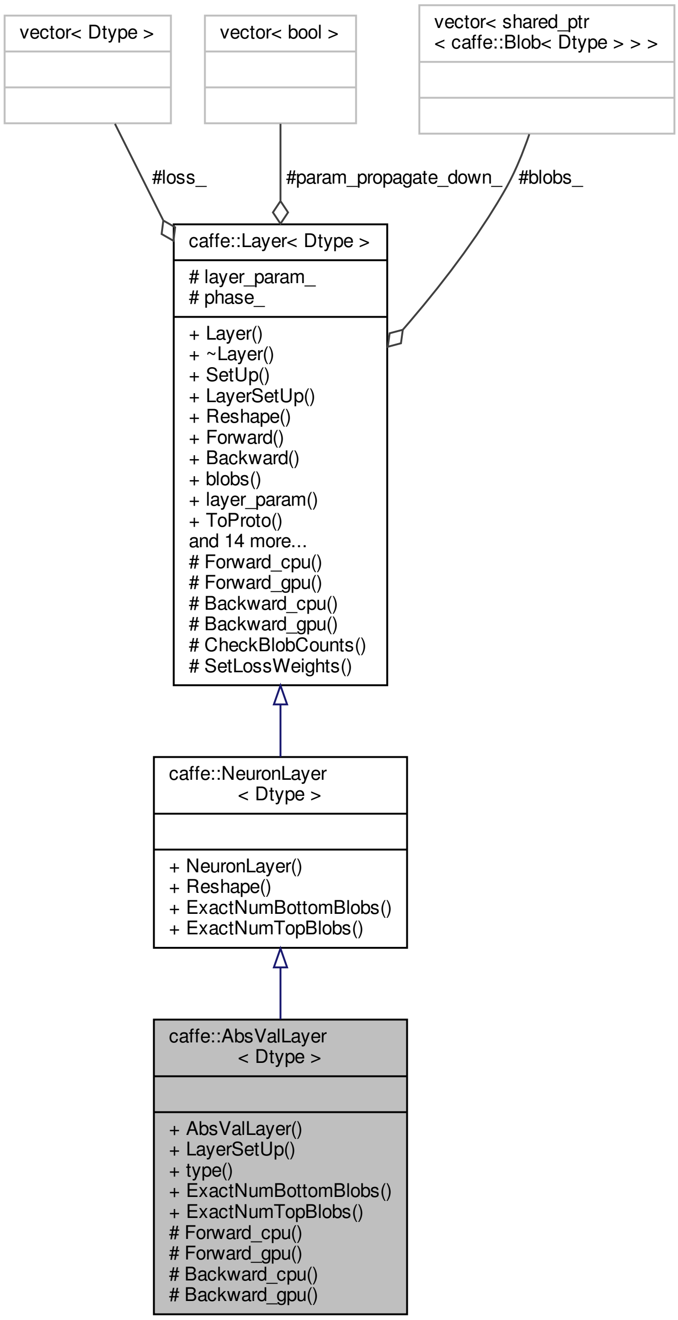
Computes $y = |x|$.

```
#include <absval_layer.hpp>
```

Inheritance diagram for caffe::AbsValLayer< Dtype >:



Collaboration diagram for caffe::AbsValLayer< Dtype >:



Public Member Functions

- **AbsValLayer** (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual const char * type () const

  Returns the layer type.

- virtual int ExactNumBottomBlobs () const

  Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.
- virtual int ExactNumTopBlobs () const

  Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

Protected Member Functions

- virtual void Forward_cpu (const vector$<$ Blob$<$ Dtype $>$ $*>$ &bottom, const vector$<$ Blob$<$ Dtype $>$ $*>$ &top)

  Computes $y = |x|$.
- virtual void Forward_gpu (const vector$<$ Blob$<$ Dtype $>$ $*>$ &bottom, const vector$<$ Blob$<$ Dtype $>$ $*>$ &top)

  Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector$<$ Blob$<$ Dtype $>$ $*>$ &top, const vector$<$ bool $>$ &propagate_down, const vector$<$ Blob$<$ Dtype $>$ $*>$ &bottom)

  Computes the error gradient w.r.t. the absolute value inputs.
- virtual void Backward_gpu (const vector$<$ Blob$<$ Dtype $>$ $*>$ &top, const vector$<$ bool $>$ &propagate_down, const vector$<$ Blob$<$ Dtype $>$ $*>$ &bottom)

  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Additional Inherited Members

5.1.1 Detailed Description

template$<$typename Dtype$>$
class caffe::AbsValLayer$<$ Dtype $>$

Computes $y = |x|$.

Parameters

bottom	input Blob vector (length 1) 1. $(N \times C \times H \times W)$ the inputs x
top	output Blob vector (length 1) 1. $(N \times C \times H \times W)$ the computed outputs $y = |x|$

5.1.2 Member Function Documentation

5.1.2.1 Backward_cpu()

```
template<typename Dtype >
void caffe::AbsValLayer< Dtype >::Backward_cpu (
```

```
            const vector< Blob< Dtype > *> & top,
            const vector< bool > & propagate_down,
            const vector< Blob< Dtype > *> & bottom )  [protected], [virtual]
```

Computes the error gradient w.r.t. the absolute value inputs.

Parameters

top	output Blob vector (length 1), providing the error gradient with respect to the outputs 1. $(N \times C \times H \times W)$ containing error gradients $\frac{\partial E}{\partial y}$ with respect to computed outputs y
propagate_down	see Layer::Backward.
bottom	input Blob vector (length 2) 1. $(N \times C \times H \times W)$ the inputs x; Backward fills their diff with gradients $\frac{\partial E}{\partial x} = \mathrm{sign}(x) \frac{\partial E}{\partial y}$ if propagate_down[0]

Implements caffe::Layer< Dtype >.

5.1.2.2 ExactNumBottomBlobs()

```
template<typename Dtype >
virtual int caffe::AbsValLayer< Dtype >::ExactNumBottomBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of bottom blobs.

Reimplemented from caffe::NeuronLayer< Dtype >.

5.1.2.3 ExactNumTopBlobs()

```
template<typename Dtype >
virtual int caffe::AbsValLayer< Dtype >::ExactNumTopBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of top blobs.

Reimplemented from caffe::NeuronLayer< Dtype >.

5.1.2.4 Forward_cpu()

```
template<typename Dtype >
void caffe::AbsValLayer< Dtype >::Forward_cpu (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [protected], [virtual]
```

Computes $y = |x|$.

Parameters

bottom	input Blob vector (length 1) 1. $(N \times C \times H \times W)$ the inputs x
top	output Blob vector (length 1) 1. $(N \times C \times H \times W)$ the computed outputs $y = \|x\|$

Implements caffe::Layer< Dtype >.

5.1.2.5 LayerSetUp()

```
template<typename Dtype >
void caffe::AbsValLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

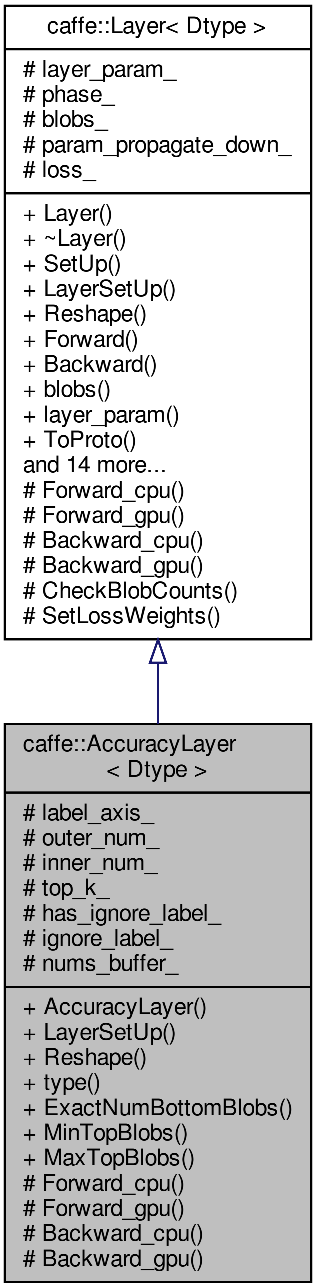
- include/caffe/layers/absval_layer.hpp
- src/caffe/layers/absval_layer.cpp

5.2 caffe::AccuracyLayer< Dtype > Class Template Reference

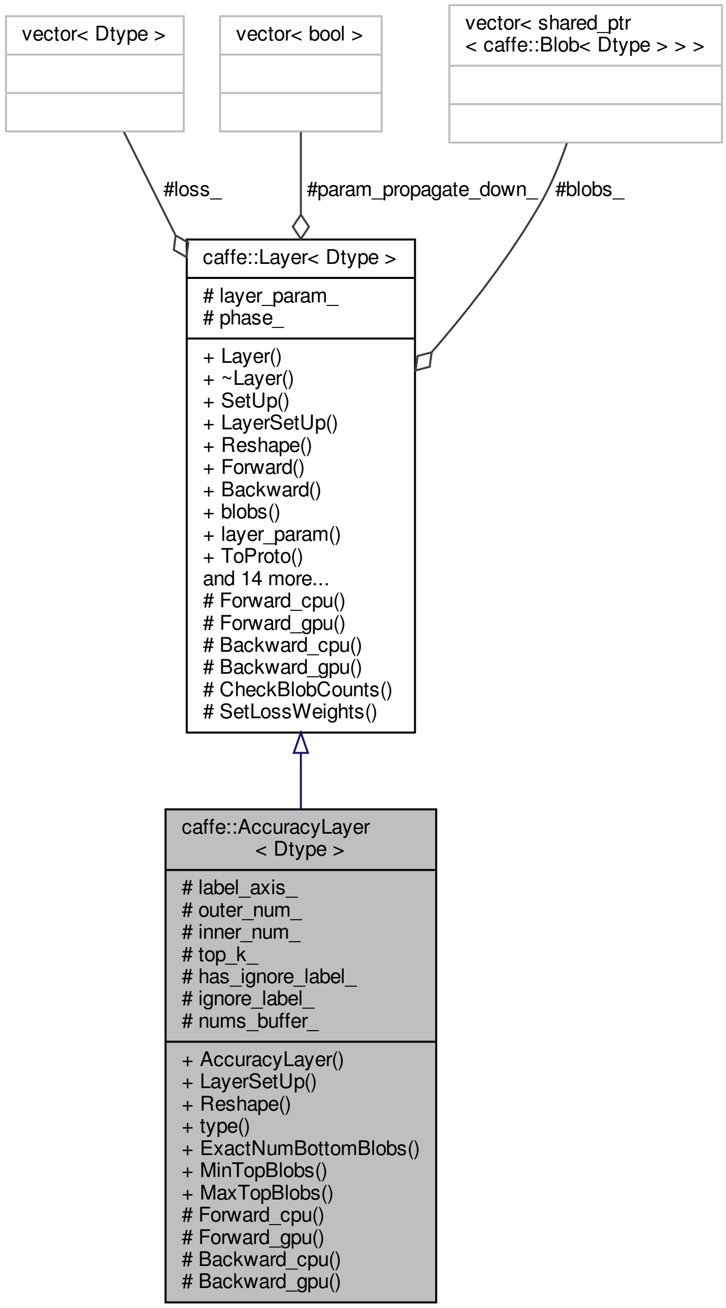
Computes the classification accuracy for a one-of-many classification task.

`#include <accuracy_layer.hpp>`

Inheritance diagram for caffe::AccuracyLayer< Dtype >:



Collaboration diagram for caffe::AccuracyLayer< Dtype >:



Public Member Functions

- AccuracyLayer (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

    Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void Reshape (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

    Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

- virtual const char * **type** () const

  Returns the layer type.
- virtual int **ExactNumBottomBlobs** () const

  Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.
- virtual int **MinTopBlobs** () const

  Returns the minimum number of top blobs required by the layer, or -1 if no minimum number is required.
- virtual int **MaxTopBlobs** () const

  Returns the maximum number of top blobs required by the layer, or -1 if no maximum number is required.

Protected Member Functions

- virtual void **Forward_cpu** (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)
- virtual void **Forward_gpu** (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void **Backward_cpu** (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Not implemented – AccuracyLayer cannot be used as a loss.
- virtual void **Backward_gpu** (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Protected Attributes

- int **label_axis_**
- int **outer_num_**
- int **inner_num_**
- int **top_k_**
- bool has_ignore_label_

  Whether to ignore instances with a certain label.
- int ignore_label_

  The label indicating that an instance should be ignored.
- Blob< Dtype > nums_buffer_

  Keeps counts of the number of samples per class.

5.2.1 Detailed Description

template<typename Dtype>
class caffe::AccuracyLayer< Dtype >

Computes the classification accuracy for a one-of-many classification task.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 AccuracyLayer()

```
template<typename Dtype >
caffe::AccuracyLayer< Dtype >::AccuracyLayer (
            const LayerParameter & param )  [inline], [explicit]
```

Parameters

param	provides AccuracyParameter accuracy_param, with AccuracyLayer options: • top_k (**optional**, default 1). Sets the maximum rank k at which a prediction is considered correct. For example, if $k = 5$, a prediction is counted correct if the correct label is among the top 5 predicted labels.

5.2.3 Member Function Documentation

5.2.3.1 ExactNumBottomBlobs()

```
template<typename Dtype >
virtual int caffe::AccuracyLayer< Dtype >::ExactNumBottomBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.2.3.2 Forward_cpu()

```
template<typename Dtype >
void caffe::AccuracyLayer< Dtype >::Forward_cpu (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [protected], [virtual]
```

Parameters

bottom	input Blob vector (length 2) 1. $(N \times C \times H \times W)$ the predictions x, a Blob with values in $[-\infty, +\infty]$ indicating the predicted score for each of the $K = CHW$ classes. Each x_n is mapped to a predicted label $\hat{l}_n$ given by its maximal index: $\hat{l}_n = \arg\max\limits_k x_{nk}$ 2. $(N \times 1 \times 1 \times 1)$ the labels l, an integer-valued Blob with values $l_n \in [0, 1, 2, ..., K-1]$ indicating the correct class label among the K classes
top	output Blob vector (length 1) 1. $(1 \times 1 \times 1 \times 1)$ the computed accuracy: $\frac{1}{N} \sum\limits_{n=1}^{N} \delta\{\hat{l}_n = l_n\}$, where $\delta\{\mathrm{condition}\} = \left\{ \begin{array}{lr} 1 & \text{if condition} \\ 0 & \text{otherwise} \end{array} \right.$

Implements caffe::Layer< Dtype >.

5.2.3.3 LayerSetUp()

```
template<typename Dtype >
void caffe::AccuracyLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

5.2.3.4 MaxTopBlobs()

```
template<typename Dtype >
virtual int caffe::AccuracyLayer< Dtype >::MaxTopBlobs ( ) const [inline], [virtual]
```

Returns the maximum number of top blobs required by the layer, or -1 if no maximum number is required.

This method should be overridden to return a non-negative value if your layer expects some maximum number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.2.3.5 MinTopBlobs()

```
template<typename Dtype >
virtual int caffe::AccuracyLayer< Dtype >::MinTopBlobs ( ) const [inline], [virtual]
```

Returns the minimum number of top blobs required by the layer, or -1 if no minimum number is required.

This method should be overridden to return a non-negative value if your layer expects some minimum number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.2.3.6 Reshape()

```
template<typename Dtype >
void caffe::AccuracyLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

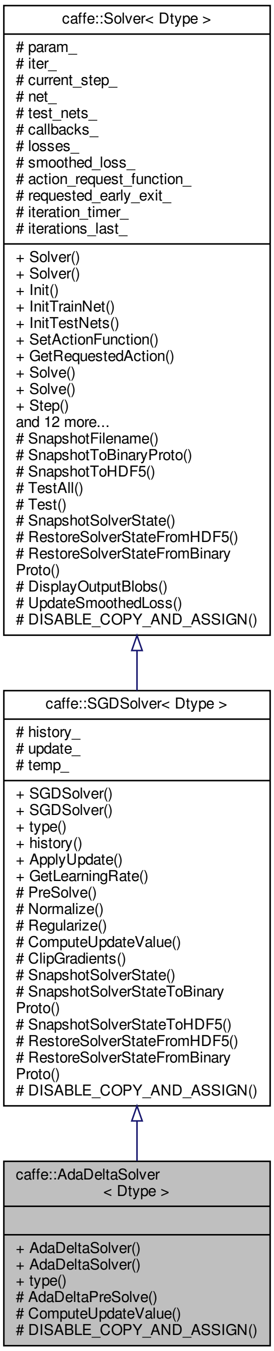
Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

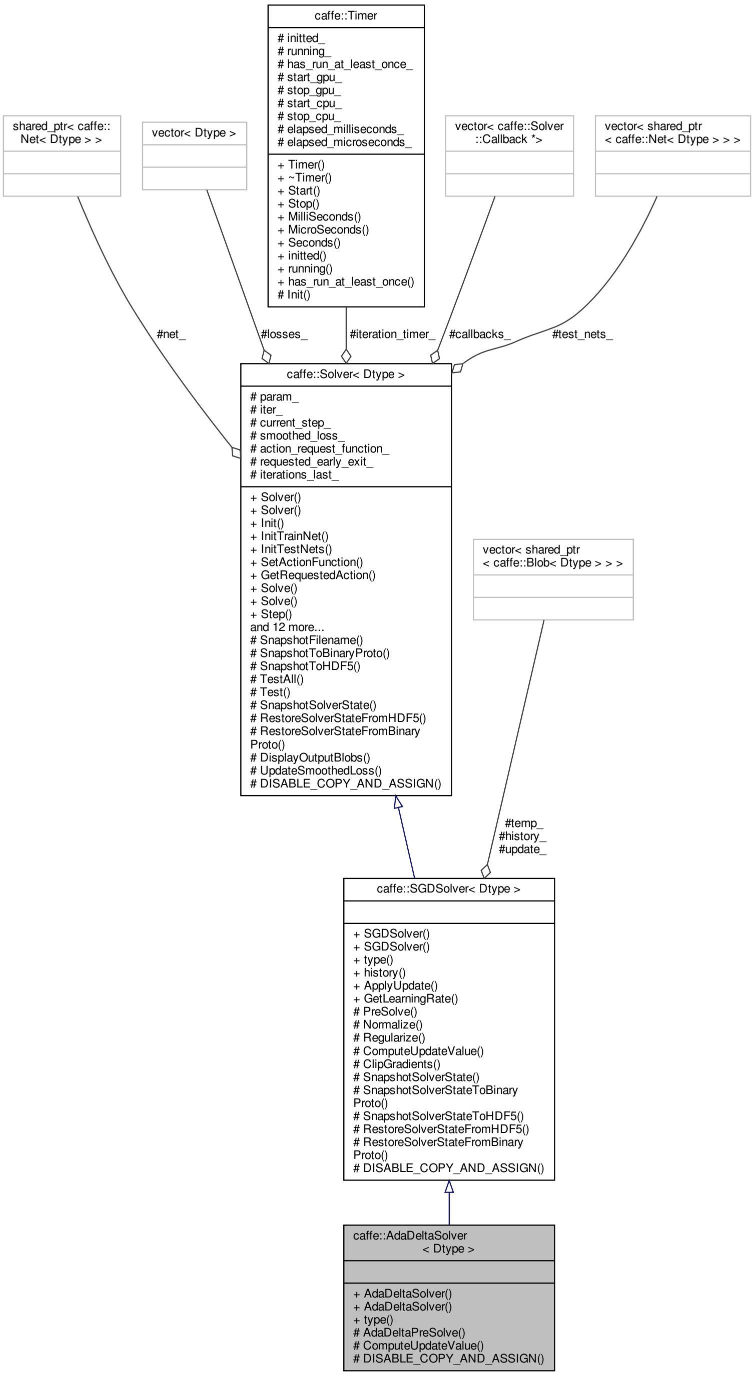
- include/caffe/layers/accuracy_layer.hpp
- src/caffe/layers/accuracy_layer.cpp

5.3 caffe::AdaDeltaSolver< Dtype > Class Template Reference

Inheritance diagram for caffe::AdaDeltaSolver< Dtype >:



Collaboration diagram for caffe::AdaDeltaSolver< Dtype >:



Public Member Functions

- **AdaDeltaSolver** (const SolverParameter &param)
- **AdaDeltaSolver** (const string &param_file)
- virtual const char ∗ type () const

  Returns the solver type.

Protected Member Functions

- void **AdaDeltaPreSolve** ()
- virtual void **ComputeUpdateValue** (int param_id, Dtype rate)
- **DISABLE_COPY_AND_ASSIGN** (AdaDeltaSolver)

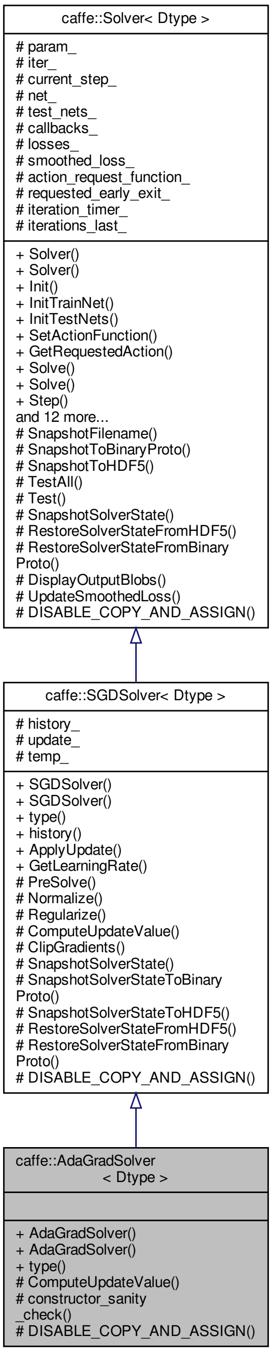
Additional Inherited Members

The documentation for this class was generated from the following files:

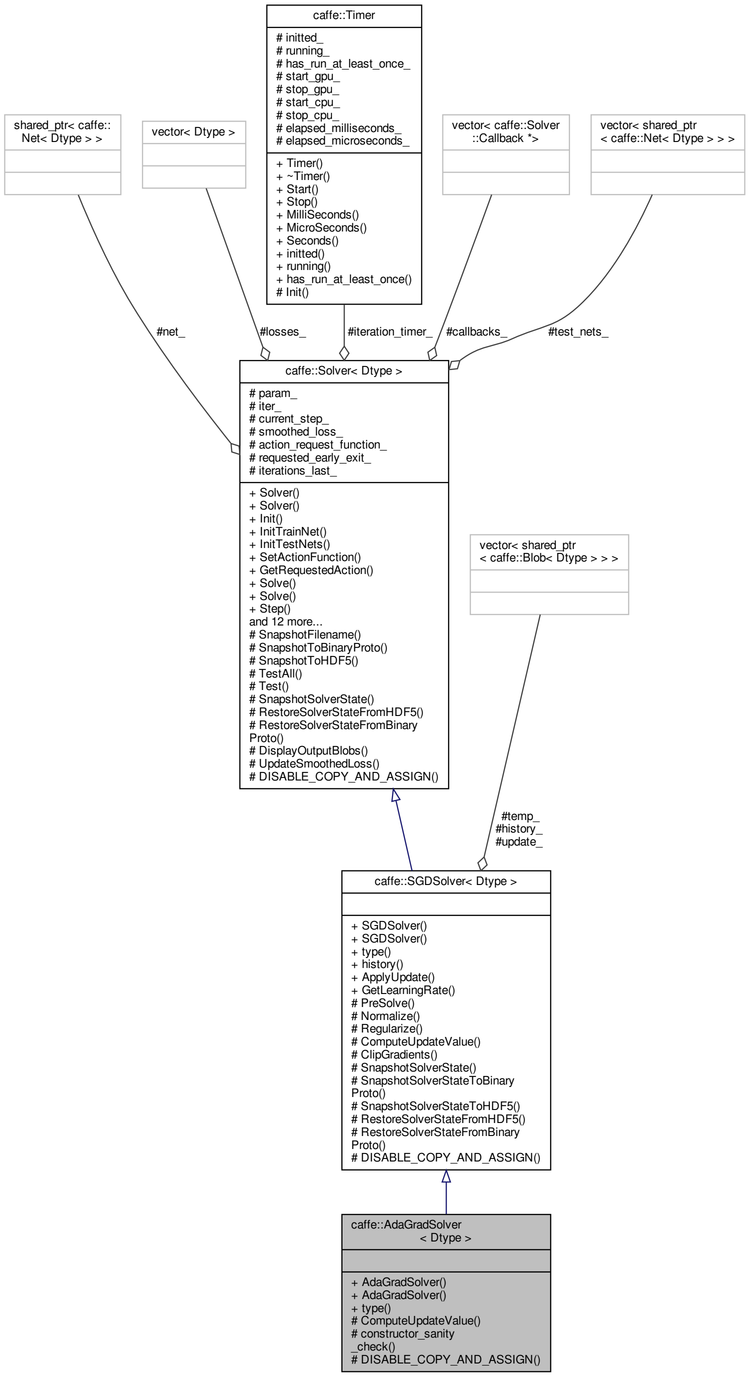
- include/caffe/sgd_solvers.hpp
- src/caffe/solvers/adadelta_solver.cpp

5.4 caffe::AdaGradSolver< Dtype > Class Template Reference

Inheritance diagram for caffe::AdaGradSolver< Dtype >:



Collaboration diagram for caffe::AdaGradSolver< Dtype >:



Public Member Functions

- **AdaGradSolver** (const SolverParameter &param)
- **AdaGradSolver** (const string &param_file)
- virtual const char * type () const

  Returns the solver type.

Protected Member Functions

- virtual void **ComputeUpdateValue** (int param_id, Dtype rate)
- void **constructor_sanity_check** ()
- **DISABLE_COPY_AND_ASSIGN** (AdaGradSolver)

Additional Inherited Members

The documentation for this class was generated from the following files:

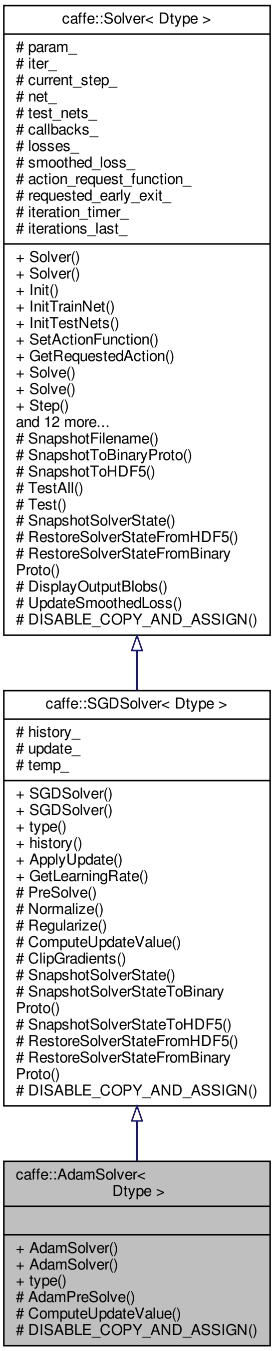
- include/caffe/sgd_solvers.hpp
- src/caffe/solvers/adagrad_solver.cpp

5.5 caffe::AdamSolver< Dtype > Class Template Reference

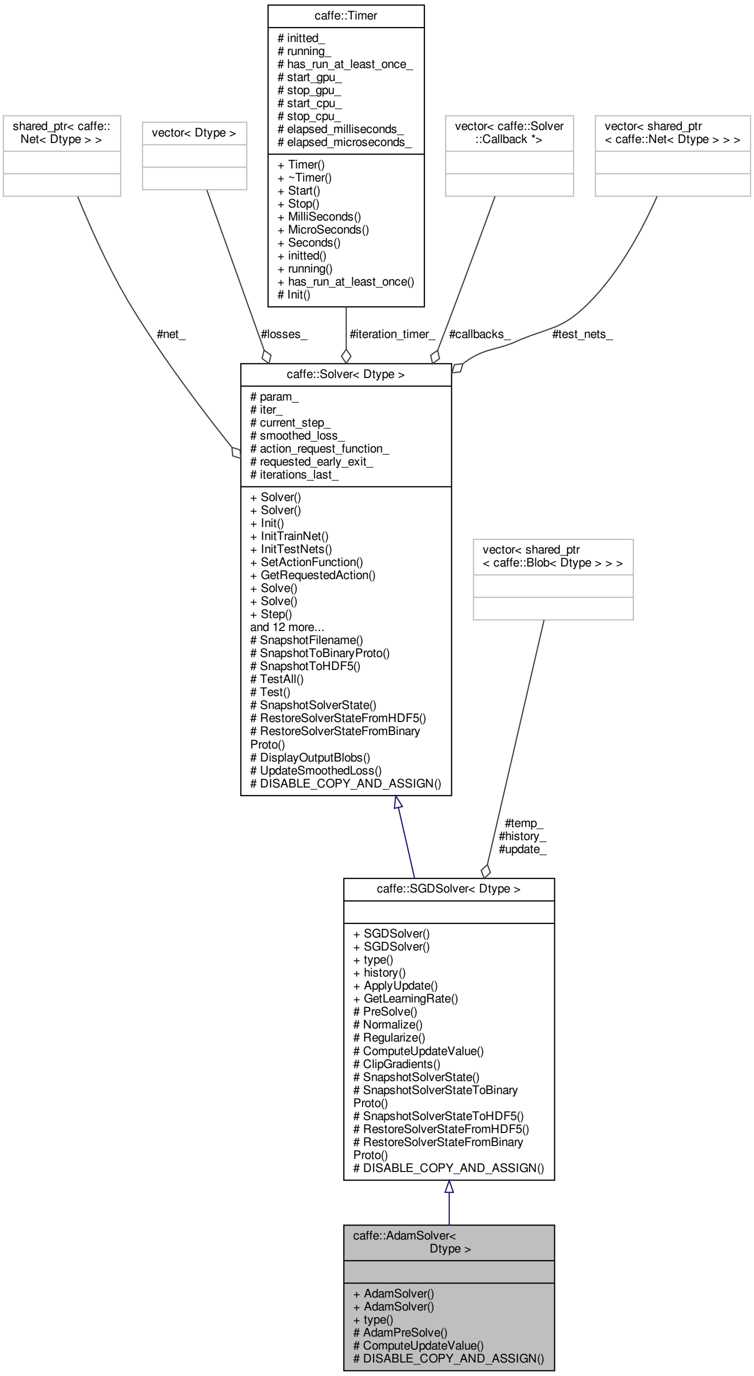
AdamSolver, an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. Described in [1].

`#include <sgd_solvers.hpp>`

Inheritance diagram for caffe::AdamSolver< Dtype >:



Collaboration diagram for caffe::AdamSolver< Dtype >:



Public Member Functions

- **AdamSolver** (const SolverParameter &param)
- **AdamSolver** (const string &param_file)
- virtual const char $*$ type () const

  Returns the solver type.

Protected Member Functions

- void **AdamPreSolve** ()
- virtual void **ComputeUpdateValue** (int param_id, Dtype rate)
- **DISABLE_COPY_AND_ASSIGN** (AdamSolver)

Additional Inherited Members

5.5.1 Detailed Description

template<typename Dtype>
class caffe::AdamSolver< Dtype >

AdamSolver, an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. Described in [1].

[1] D. P. Kingma and J. L. Ba, "ADAM: A Method for Stochastic Optimization." arXiv preprint arXiv:1412.6980v8 (2014).

The documentation for this class was generated from the following files:

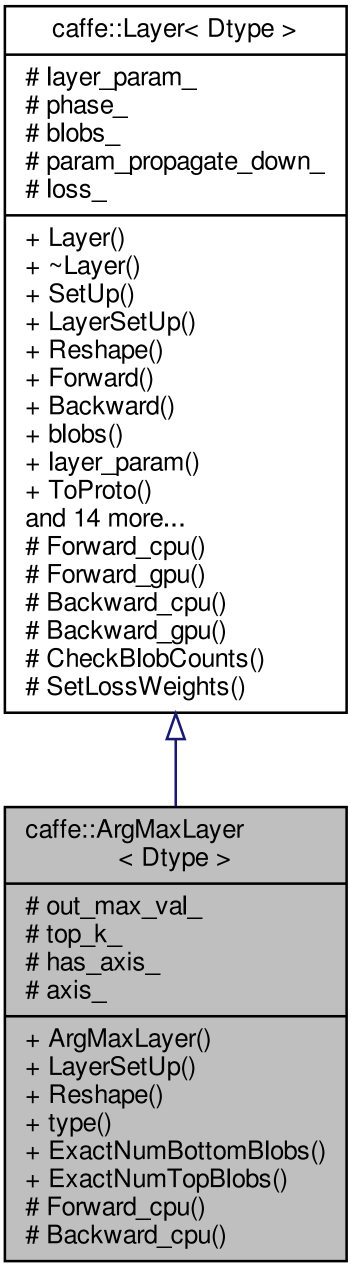
- include/caffe/sgd_solvers.hpp
- src/caffe/solvers/adam_solver.cpp

5.6 caffe::ArgMaxLayer< Dtype > Class Template Reference

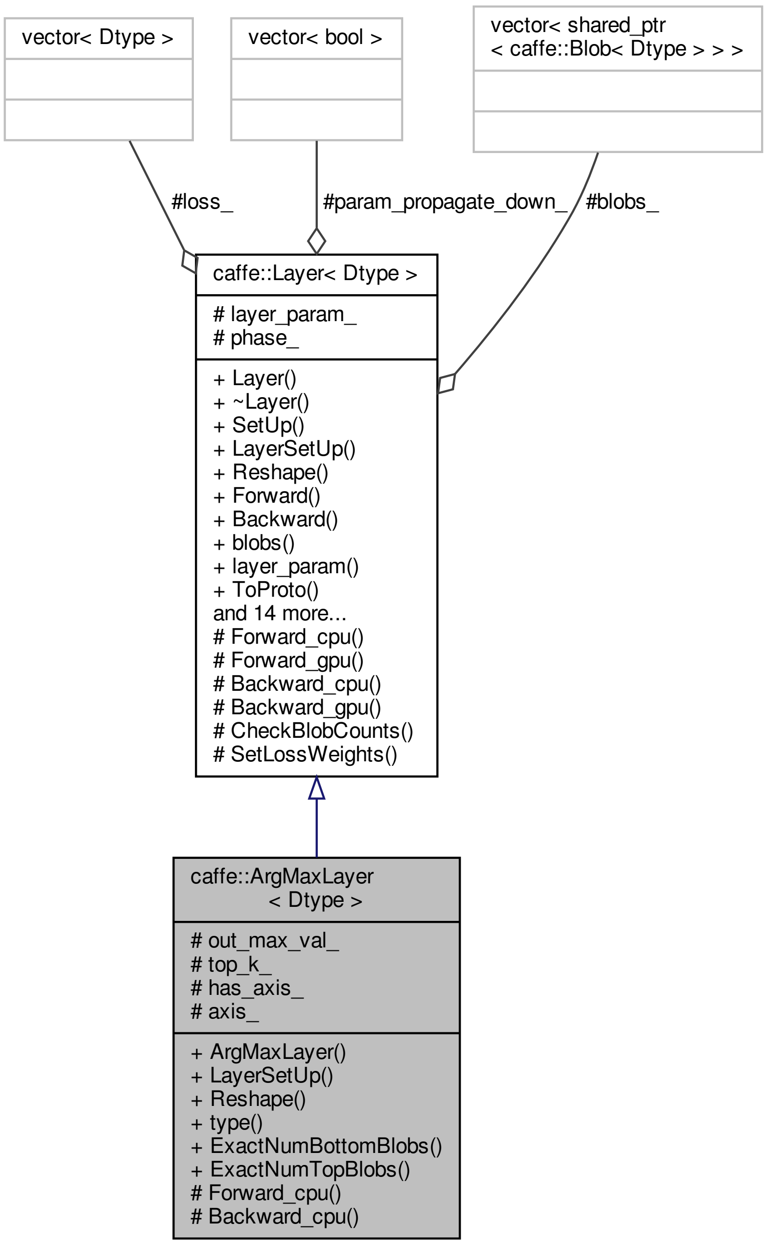
Compute the index of the K max values for each datum across all dimensions $(C \times H \times W)$.

```
#include <argmax_layer.hpp>
```

Inheritance diagram for caffe::ArgMaxLayer< Dtype >:



Collaboration diagram for caffe::ArgMaxLayer< Dtype >:



Public Member Functions

- ArgMaxLayer (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void Reshape (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.
- virtual const char ∗ type () const

Returns the layer type.
- virtual int ExactNumBottomBlobs () const

  Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.
- virtual int ExactNumTopBlobs () const

  Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Not implemented (non-differentiable function)

Protected Attributes

- bool **out_max_val_**
- size_t **top_k_**
- bool **has_axis_**
- int **axis_**

5.6.1 Detailed Description

template< typename Dtype >
class caffe::ArgMaxLayer< Dtype >

Compute the index of the K max values for each datum across all dimensions $(C \times H \times W)$.

Intended for use after a classification layer to produce a prediction. If parameter out_max_val is set to true, output is a vector of pairs (max_ind, max_val) for each image. The axis parameter specifies an axis along which to maximise.

NOTE: does not implement Backwards operation.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 ArgMaxLayer()

```
template<typename Dtype >
caffe::ArgMaxLayer< Dtype >::ArgMaxLayer (
            const LayerParameter & param )  [inline], [explicit]
```

Parameters

param	provides ArgMaxParameter argmax_param, with ArgMaxLayer options: • top_k (**optional** uint, default 1). the number K of maximal items to output. • out_max_val (**optional** bool, default false). if set, output a vector of pairs (max_ind, max_val) unless axis is set then output max_val along the specified axis. • axis (**optional** int). if set, maximise along the specified axis else maximise the flattened trailing dimensions for each index of the first / num dimension.

5.6.3 Member Function Documentation

5.6.3.1 ExactNumBottomBlobs()

```
template<typename Dtype >
virtual int caffe::ArgMaxLayer< Dtype >::ExactNumBottomBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.6.3.2 ExactNumTopBlobs()

```
template<typename Dtype >
virtual int caffe::ArgMaxLayer< Dtype >::ExactNumTopBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.6.3.3 Forward_cpu()

```
template<typename Dtype >
void caffe::ArgMaxLayer< Dtype >::Forward_cpu (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [protected], [virtual]
```

Parameters

bottom	input Blob vector (length 1) 1. $(N \times C \times H \times W)$ the inputs x
top	output Blob vector (length 1) 1. $(N \times 1 \times K)$ or, if out_max_val $(N \times 2 \times K)$ unless axis set than e.g. $(N \times K \times H \times W)$ if axis == 1 the computed outputs $y_n = \arg\max_i x_{ni}$ (for $K = 1$).

Implements caffe::Layer< Dtype >.

5.6.3.4 LayerSetUp()

```
template<typename Dtype >
void caffe::ArgMaxLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

5.6.3.5 Reshape()

```
template<typename Dtype >
void caffe::ArgMaxLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

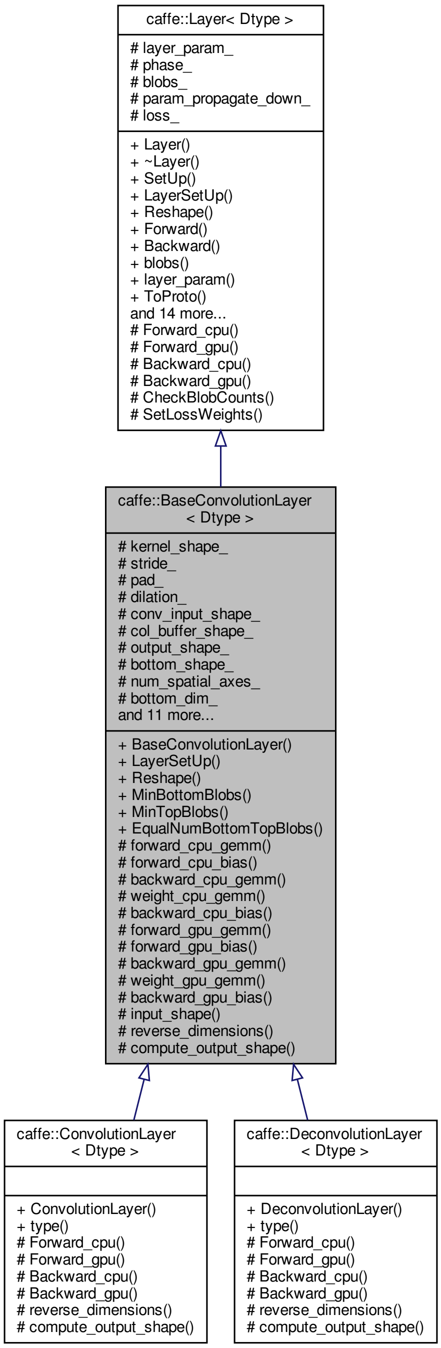
- include/caffe/layers/argmax_layer.hpp
- src/caffe/layers/argmax_layer.cpp

5.7 caffe::BaseConvolutionLayer< Dtype > Class Template Reference

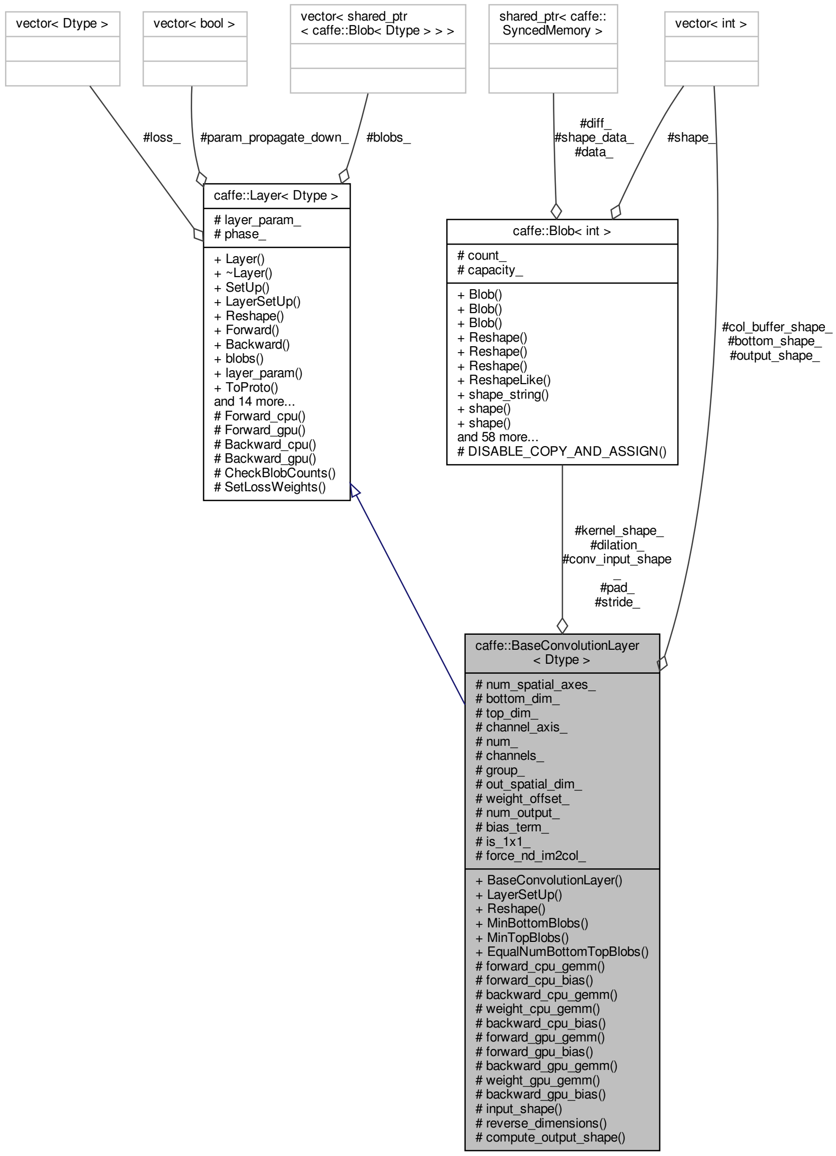
Abstract base class that factors out the BLAS code common to ConvolutionLayer and DeconvolutionLayer.

`#include <base_conv_layer.hpp>`

Inheritance diagram for caffe::BaseConvolutionLayer< Dtype >:



Collaboration diagram for caffe::BaseConvolutionLayer< Dtype >:



Public Member Functions

- **BaseConvolutionLayer** (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void Reshape (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.
- virtual int MinBottomBlobs () const

  Returns the minimum number of bottom blobs required by the layer, or -1 if no minimum number is required.
- virtual int MinTopBlobs () const

  Returns the minimum number of top blobs required by the layer, or -1 if no minimum number is required.
- virtual bool EqualNumBottomTopBlobs () const

  Returns true if the layer requires an equal number of bottom and top blobs.

Protected Member Functions

- void **forward_cpu_gemm** (const Dtype *input, const Dtype *weights, Dtype *output, bool skip_im2col=false)
- void **forward_cpu_bias** (Dtype *output, const Dtype *bias)
- void **backward_cpu_gemm** (const Dtype *input, const Dtype *weights, Dtype *output)
- void **weight_cpu_gemm** (const Dtype *input, const Dtype *output, Dtype *weights)
- void **backward_cpu_bias** (Dtype *bias, const Dtype *input)
- void **forward_gpu_gemm** (const Dtype *col_input, const Dtype *weights, Dtype *output, bool skip_↩ im2col=false)
- void **forward_gpu_bias** (Dtype *output, const Dtype *bias)
- void **backward_gpu_gemm** (const Dtype *input, const Dtype *weights, Dtype *col_output)
- void **weight_gpu_gemm** (const Dtype *col_input, const Dtype *output, Dtype *weights)
- void **backward_gpu_bias** (Dtype *bias, const Dtype *input)
- int input_shape (int i)

  The spatial dimensions of the input.
- virtual bool **reverse_dimensions** ()=0
- virtual void **compute_output_shape** ()=0

Protected Attributes

- Blob< int > kernel_shape_

  The spatial dimensions of a filter kernel.
- Blob< int > stride_

  The spatial dimensions of the stride.
- Blob< int > pad_

  The spatial dimensions of the padding.
- Blob< int > dilation_

  The spatial dimensions of the dilation.
- Blob< int > conv_input_shape_

  The spatial dimensions of the convolution input.
- vector< int > col_buffer_shape_

  The spatial dimensions of the col_buffer.
- vector< int > output_shape_

  The spatial dimensions of the output.
- const vector< int > * **bottom_shape_**
- int **num_spatial_axes_**
- int **bottom_dim_**
- int **top_dim_**
- int **channel_axis_**
- int **num_**
- int **channels_**
- int **group_**
- int **out_spatial_dim_**
- int **weight_offset_**
- int **num_output_**
- bool **bias_term_**
- bool **is_1x1_**
- bool **force_nd_im2col_**

5.7.1 Detailed Description

template<typename Dtype>
class caffe::BaseConvolutionLayer< Dtype >

Abstract base class that factors out the BLAS code common to ConvolutionLayer and DeconvolutionLayer.

5.7.2 Member Function Documentation

5.7.2.1 EqualNumBottomTopBlobs()

```
template<typename Dtype >
virtual bool caffe::BaseConvolutionLayer< Dtype >::EqualNumBottomTopBlobs ( ) const  [inline],
[virtual]
```

Returns true if the layer requires an equal number of bottom and top blobs.

This method should be overridden to return true if your layer expects an equal number of bottom and top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.7.2.2 LayerSetUp()

```
template<typename Dtype >
void caffe::BaseConvolutionLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

5.7.2.3 MinBottomBlobs()

```
template<typename Dtype >
virtual int caffe::BaseConvolutionLayer< Dtype >::MinBottomBlobs ( ) const [inline], [virtual]
```

Returns the minimum number of bottom blobs required by the layer, or -1 if no minimum number is required.

This method should be overridden to return a non-negative value if your layer expects some minimum number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.7.2.4 MinTopBlobs()

```
template<typename Dtype >
virtual int caffe::BaseConvolutionLayer< Dtype >::MinTopBlobs ( ) const [inline], [virtual]
```

Returns the minimum number of top blobs required by the layer, or -1 if no minimum number is required.

This method should be overridden to return a non-negative value if your layer expects some minimum number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.7.2.5 Reshape()

```
template<typename Dtype >
void caffe::BaseConvolutionLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

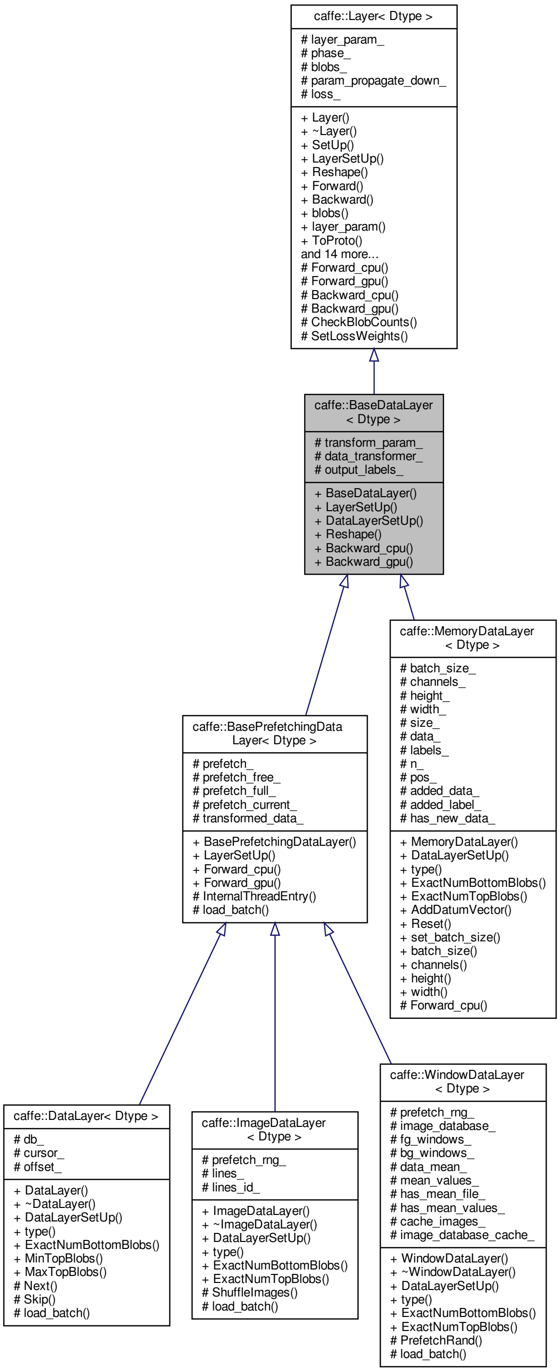
- include/caffe/layers/base_conv_layer.hpp
- src/caffe/layers/base_conv_layer.cpp

5.8 caffe::BaseDataLayer< Dtype > Class Template Reference

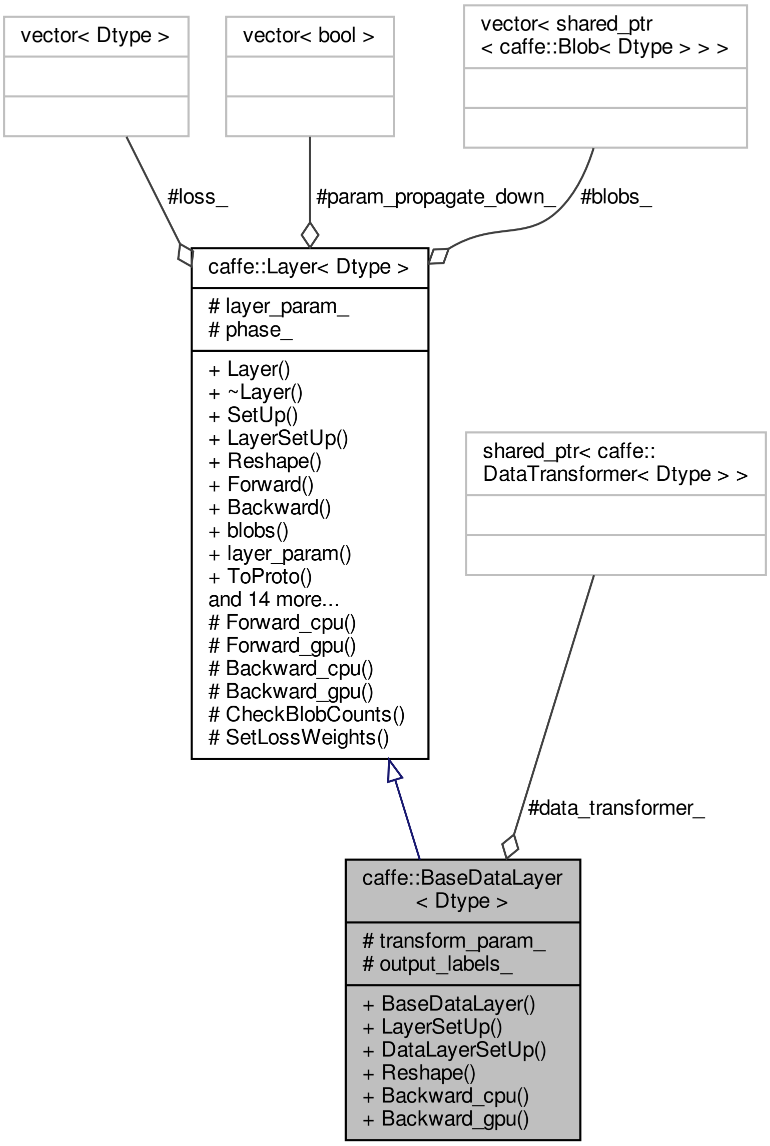
Provides base for data layers that feed blobs to the Net.

```
#include <base_data_layer.hpp>
```

Inheritance diagram for caffe::BaseDataLayer< Dtype >:



Collaboration diagram for caffe::BaseDataLayer< Dtype >:



Public Member Functions

- **BaseDataLayer** (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void **DataLayerSetUp** (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)
- virtual void Reshape (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.
- virtual void Backward_cpu (const vector< Blob< Dtype > ∗> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > ∗> &bottom)

Using the CPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true.

- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Protected Attributes

- TransformationParameter **transform_param_**
- shared_ptr< DataTransformer< Dtype > > **data_transformer_**
- bool **output_labels_**

Additional Inherited Members

5.8.1 Detailed Description

template<typename Dtype>
class caffe::BaseDataLayer< Dtype >

Provides base for data layers that feed blobs to the Net.

TODO(dox): thorough documentation for Forward and proto params.

5.8.2 Member Function Documentation

5.8.2.1 LayerSetUp()

```
template<typename Dtype >
void caffe::BaseDataLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

Reimplemented in caffe::BasePrefetchingDataLayer< Dtype >.

5.8.2.2 Reshape()

```
template<typename Dtype >
virtual void caffe::BaseDataLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [inline], [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

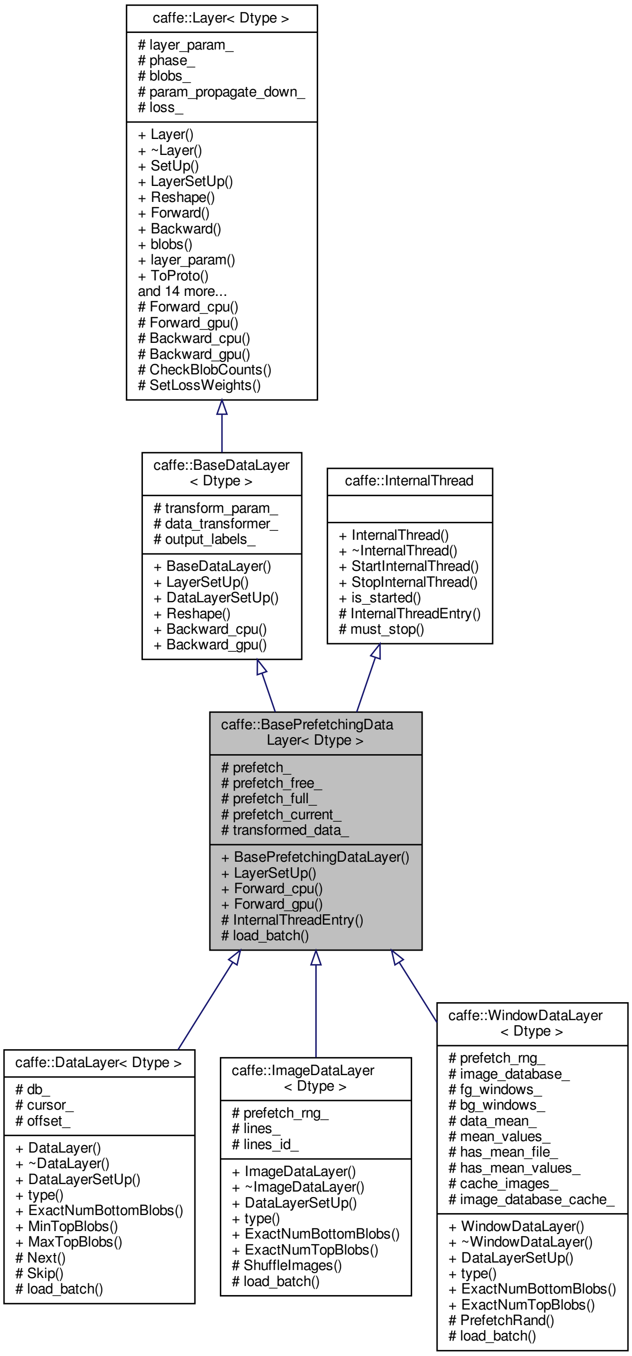
Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

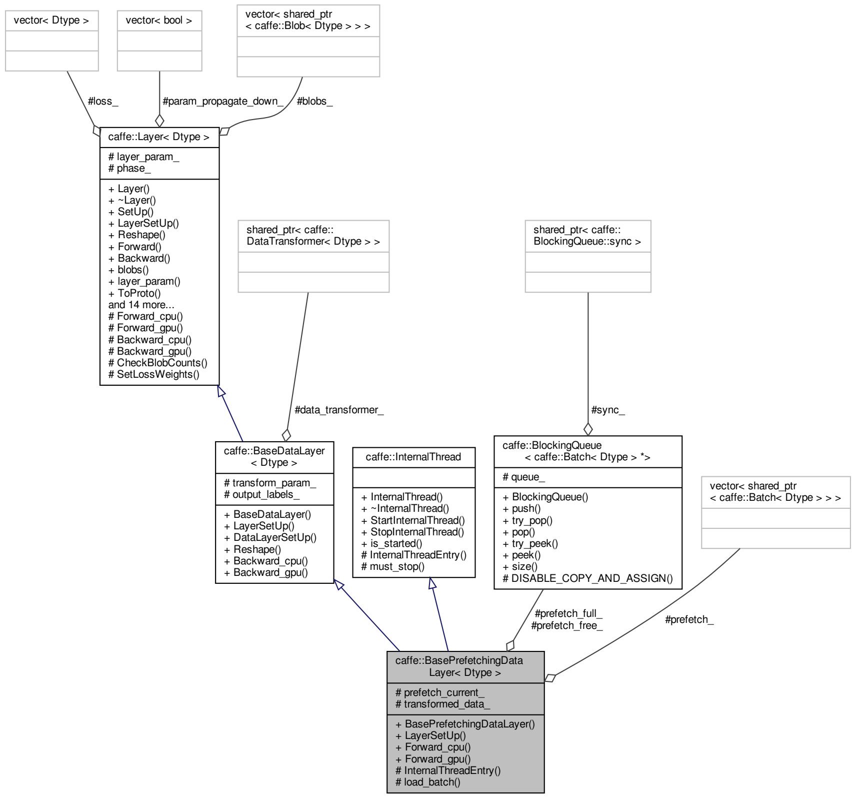
- include/caffe/layers/base_data_layer.hpp
- src/caffe/layers/base_data_layer.cpp

5.9 caffe::BasePrefetchingDataLayer< Dtype > Class Template Reference

Inheritance diagram for caffe::BasePrefetchingDataLayer< Dtype >:



Collaboration diagram for caffe::BasePrefetchingDataLayer< Dtype >:



Public Member Functions

- **BasePrefetchingDataLayer** (const LayerParameter &param)
- void LayerSetUp (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the CPU device, compute the layer output.
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.

Protected Member Functions

- virtual void **InternalThreadEntry** ()
- virtual void **load_batch** (Batch< Dtype > *batch)=0

Protected Attributes

- vector< shared_ptr< Batch< Dtype > > > **prefetch_**
- BlockingQueue< Batch< Dtype > * > **prefetch_free_**
- BlockingQueue< Batch< Dtype > * > **prefetch_full_**
- Batch< Dtype > * **prefetch_current_**
- Blob< Dtype > **transformed_data_**

5.9.1 Member Function Documentation

5.9.1.1 LayerSetUp()

```
template<typename Dtype >
void caffe::BasePrefetchingDataLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

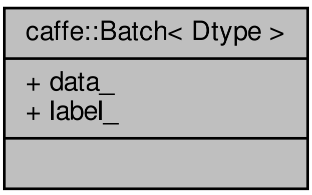
Reimplemented from caffe::BaseDataLayer< Dtype >.

The documentation for this class was generated from the following files:

- include/caffe/layers/base_data_layer.hpp
- src/caffe/layers/base_data_layer.cpp

5.10 caffe::Batch< Dtype > Class Template Reference

Collaboration diagram for caffe::Batch< Dtype >:



Public Attributes

- Blob< Dtype > **data_**
- Blob< Dtype > **label_**

The documentation for this class was generated from the following file:

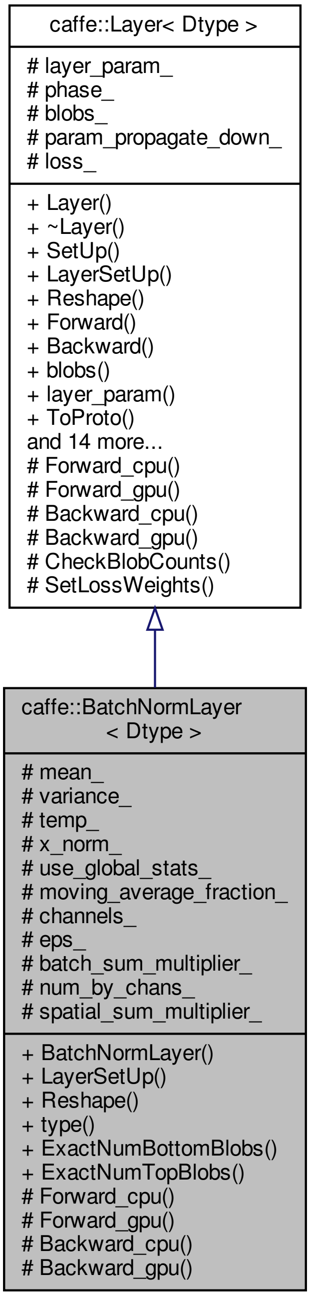
- include/caffe/layers/base_data_layer.hpp

5.11 caffe::BatchNormLayer< Dtype > Class Template Reference

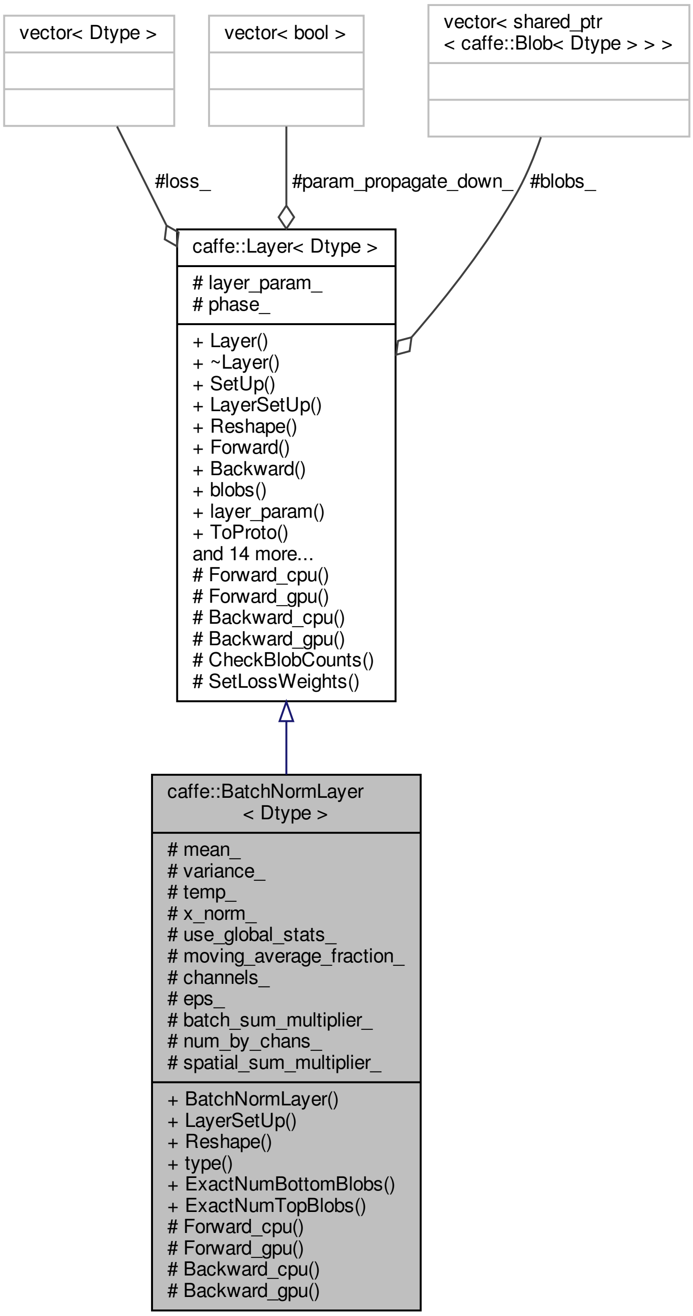
Normalizes the input to have 0-mean and/or unit (1) variance across the batch.

`#include <batch_norm_layer.hpp>`

Inheritance diagram for caffe::BatchNormLayer< Dtype >:



Collaboration diagram for caffe::BatchNormLayer< Dtype >:



Public Member Functions

- **BatchNormLayer** (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

    Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void Reshape (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

    Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

- virtual const char * type () const

  Returns the layer type.
- virtual int ExactNumBottomBlobs () const

  Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.
- virtual int ExactNumTopBlobs () const

  Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the CPU device, compute the layer output.
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the CPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Protected Attributes

- Blob< Dtype > **mean_**
- Blob< Dtype > **variance_**
- Blob< Dtype > **temp_**
- Blob< Dtype > **x_norm_**
- bool **use_global_stats_**
- Dtype **moving_average_fraction_**
- int **channels_**
- Dtype **eps_**
- Blob< Dtype > **batch_sum_multiplier_**
- Blob< Dtype > **num_by_chans_**
- Blob< Dtype > **spatial_sum_multiplier_**

5.11.1 Detailed Description

template<typename Dtype>
class caffe::BatchNormLayer< Dtype >

Normalizes the input to have 0-mean and/or unit (1) variance across the batch.

This layer computes Batch Normalization as described in [1]. For each channel in the data (i.e. axis 1), it subtracts the mean and divides by the variance, where both statistics are computed across both spatial dimensions and across the different examples in the batch.

By default, during training time, the network is computing global mean/variance statistics via a running average, which is then used at test time to allow deterministic outputs for each input. You can manually toggle whether the network is accumulating or using the statistics via the use_global_stats option. For reference, these statistics are kept in the layer's three blobs: (0) mean, (1) variance, and (2) moving average factor.

Note that the original paper also included a per-channel learned bias and scaling factor. To implement this in Caffe, define a `ScaleLayer` configured with `bias_term: true` after each `BatchNormLayer` to handle both the bias and scaling factor.

[1] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." arXiv preprint arXiv:1502.03167 (2015).

TODO(dox): thorough documentation for Forward, Backward, and proto params.

5.11.2 Member Function Documentation

5.11.2.1 ExactNumBottomBlobs()

```
template<typename Dtype >
virtual int caffe::BatchNormLayer< Dtype >::ExactNumBottomBlobs ( ) const [inline], [virtual]
```

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.11.2.2 ExactNumTopBlobs()

```
template<typename Dtype >
virtual int caffe::BatchNormLayer< Dtype >::ExactNumTopBlobs ( ) const [inline], [virtual]
```

Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.11.2.3 LayerSetUp()

```
template<typename Dtype >
void caffe::BatchNormLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

5.11.2.4 Reshape()

```
template<typename Dtype >
void caffe::BatchNormLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

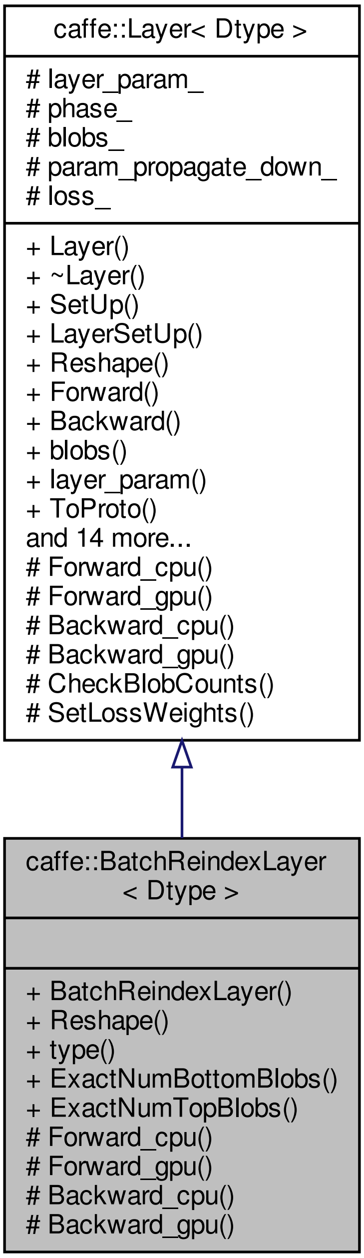
- include/caffe/layers/batch_norm_layer.hpp
- src/caffe/layers/batch_norm_layer.cpp

5.12 caffe::BatchReindexLayer< Dtype > Class Template Reference

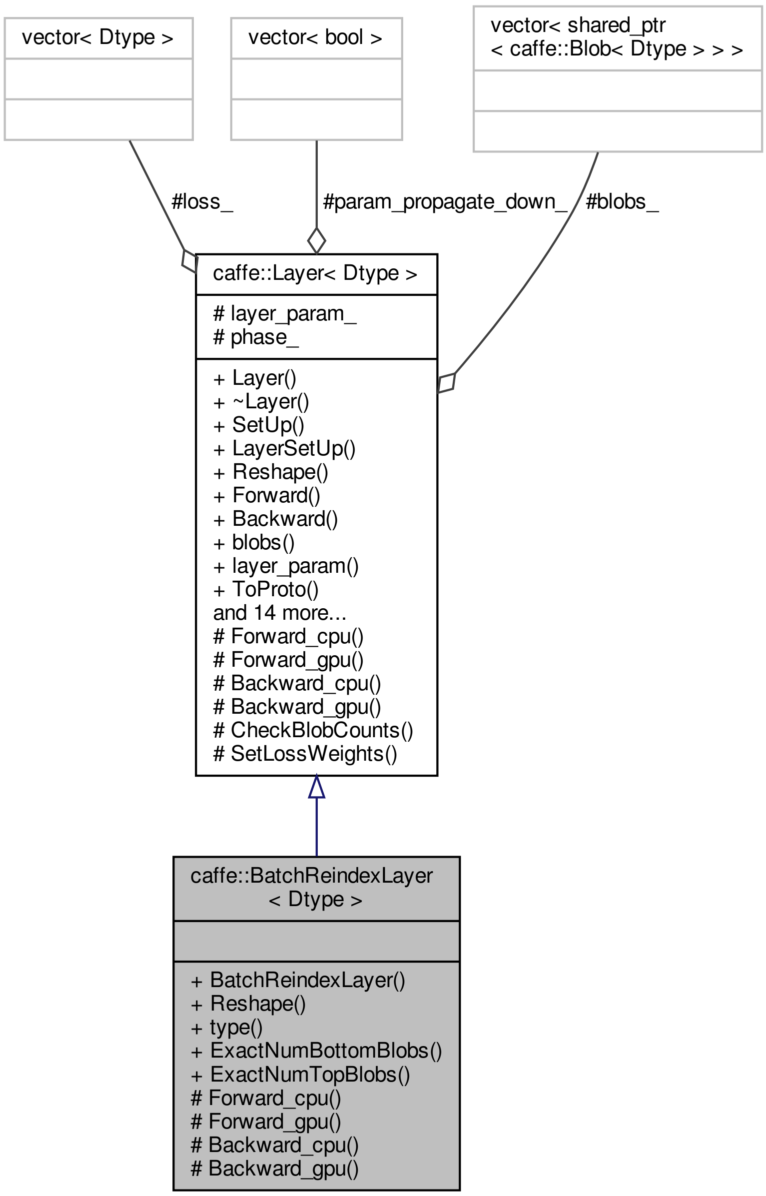
Index into the input blob along its first axis.

```
#include <batch_reindex_layer.hpp>
```

Inheritance diagram for caffe::BatchReindexLayer< Dtype >:



Collaboration diagram for caffe::BatchReindexLayer< Dtype >:



Public Member Functions

- **BatchReindexLayer** (const LayerParameter &param)
- virtual void Reshape (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

    Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.
- virtual const char ∗ type () const

    Returns the layer type.
- virtual int ExactNumBottomBlobs () const

    Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.
- virtual int ExactNumTopBlobs () const

    Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Computes the error gradient w.r.t. the reordered input.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Additional Inherited Members

5.12.1 Detailed Description

template<typename Dtype>
class caffe::BatchReindexLayer< Dtype >

Index into the input blob along its first axis.

This layer can be used to select, reorder, and even replicate examples in a batch. The second blob is cast to int and treated as an index into the first axis of the first blob.

5.12.2 Member Function Documentation

5.12.2.1 Backward_cpu()

```
template<typename Dtype >
void caffe::BatchReindexLayer< Dtype >::Backward_cpu (
            const vector< Blob< Dtype > *> & top,
            const vector< bool > & propagate_down,
            const vector< Blob< Dtype > *> & bottom )  [protected], [virtual]
```

Computes the error gradient w.r.t. the reordered input.

Parameters

top	output Blob vector (length 1), providing the error gradient with respect to the outputs 1. $(M \times ...)$: containing error gradients $\frac{\partial E}{\partial y}$ with respect to concatenated outputs y
propagate_down	see Layer::Backward.
bottom	input Blob vector (length 2): • $\frac{\partial E}{\partial y}$ is de-indexed (summing where required) back to the input x_1
	• This layer cannot backprop to x_2, i.e. propagate_down[1] must be false.

Implements caffe::Layer< Dtype >.

5.12.2.2 ExactNumBottomBlobs()

```
template<typename Dtype >
virtual int caffe::BatchReindexLayer< Dtype >::ExactNumBottomBlobs ( ) const [inline], [virtual]
```

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.12.2.3 ExactNumTopBlobs()

```
template<typename Dtype >
virtual int caffe::BatchReindexLayer< Dtype >::ExactNumTopBlobs ( ) const [inline], [virtual]
```

Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.12.2.4 Forward_cpu()

```
template<typename Dtype >
void caffe::BatchReindexLayer< Dtype >::Forward_cpu (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [protected], [virtual]
```

Parameters

bottom	input Blob vector (length 2+) 1. $(N \times ...)$ the inputs x_1 2. (M) the inputs x_2
top	output Blob vector (length 1) 1. $(M \times ...)$: the reindexed array $y = x_1[x_2]$

Implements caffe::Layer< Dtype >.

5.12.2.5 Reshape()

```
template<typename Dtype >
void caffe::BatchReindexLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

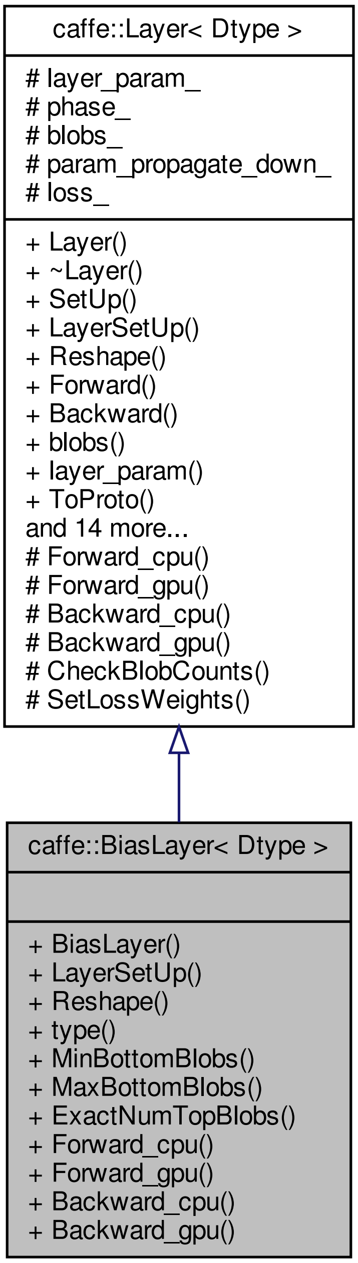
- include/caffe/layers/batch_reindex_layer.hpp
- src/caffe/layers/batch_reindex_layer.cpp

5.13 caffe::BiasLayer< Dtype > Class Template Reference

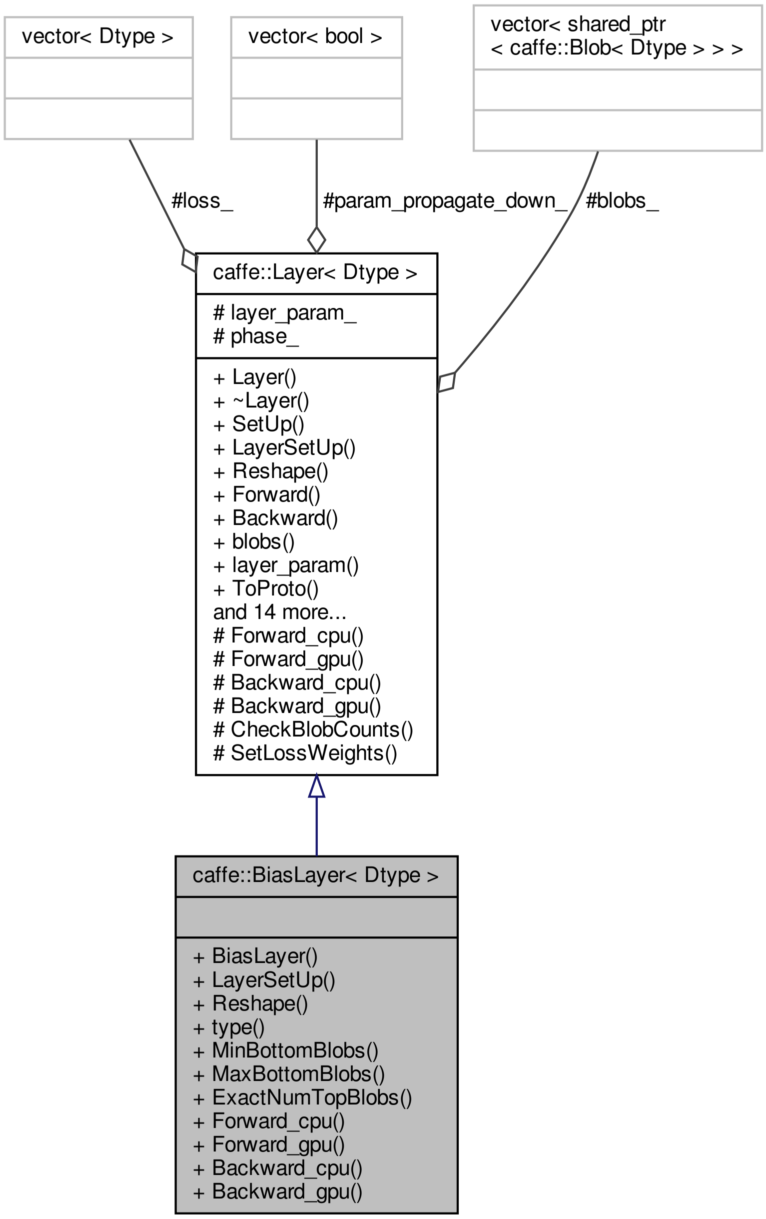
Computes a sum of two input Blobs, with the shape of the latter Blob "broadcast" to match the shape of the former. Equivalent to tiling the latter Blob, then computing the elementwise sum.

```
#include <bias_layer.hpp>
```

Inheritance diagram for caffe::BiasLayer< Dtype >:



Collaboration diagram for caffe::BiasLayer< Dtype >:



Public Member Functions

- **BiasLayer** (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

    Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void Reshape (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

    Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.
- virtual const char ∗ type () const

    Returns the layer type.

- virtual int MinBottomBlobs () const

  Returns the minimum number of bottom blobs required by the layer, or -1 if no minimum number is required.
- virtual int MaxBottomBlobs () const

  Returns the maximum number of bottom blobs required by the layer, or -1 if no maximum number is required.
- virtual int ExactNumTopBlobs () const

  Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.
- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the CPU device, compute the layer output.
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the CPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Additional Inherited Members

5.13.1 Detailed Description

template<typename Dtype>
class caffe::BiasLayer< Dtype >

Computes a sum of two input Blobs, with the shape of the latter Blob "broadcast" to match the shape of the former. Equivalent to tiling the latter Blob, then computing the elementwise sum.

The second input may be omitted, in which case it's learned as a parameter of the layer. Note: in case bias and scaling are desired, both operations can be handled by `ScaleLayer` configured with `bias_term: true`.

5.13.2 Member Function Documentation

5.13.2.1 ExactNumTopBlobs()

```
template<typename Dtype >
virtual int caffe::BiasLayer< Dtype >::ExactNumTopBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.13.2.2 LayerSetUp()

```
template<typename Dtype >
void caffe::BiasLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

5.13.2.3 MaxBottomBlobs()

```
template<typename Dtype >
virtual int caffe::BiasLayer< Dtype >::MaxBottomBlobs ( ) const  [inline], [virtual]
```

Returns the maximum number of bottom blobs required by the layer, or -1 if no maximum number is required.

This method should be overridden to return a non-negative value if your layer expects some maximum number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.13.2.4 MinBottomBlobs()

```
template<typename Dtype >
virtual int caffe::BiasLayer< Dtype >::MinBottomBlobs ( ) const  [inline], [virtual]
```

Returns the minimum number of bottom blobs required by the layer, or -1 if no minimum number is required.

This method should be overridden to return a non-negative value if your layer expects some minimum number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.13.2.5 Reshape()

```
template<typename Dtype >
void caffe::BiasLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

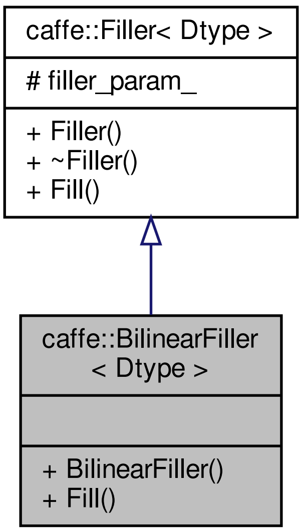
- include/caffe/layers/bias_layer.hpp
- src/caffe/layers/bias_layer.cpp

5.14 caffe::BilinearFiller< Dtype > Class Template Reference

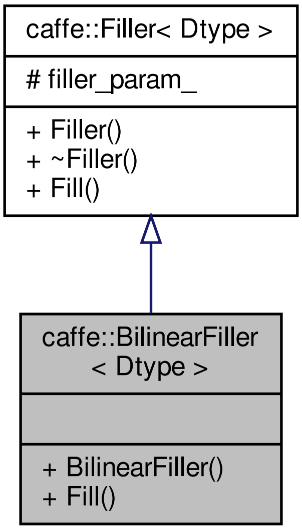
Fills a Blob with coefficients for bilinear interpolation.

```
#include <filler.hpp>
```

Inheritance diagram for caffe::BilinearFiller< Dtype >:



Collaboration diagram for caffe::BilinearFiller< Dtype >:



Public Member Functions

- **BilinearFiller** (const FillerParameter &param)
- virtual void **Fill** (Blob< Dtype > *blob)

Additional Inherited Members

5.14.1 Detailed Description

template<typename Dtype>
class caffe::BilinearFiller< Dtype >

Fills a Blob with coefficients for bilinear interpolation.

A common use case is with the DeconvolutionLayer acting as upsampling. You can upsample a feature map with shape of (B, C, H, W) by any integer factor using the following proto.

```
layer {
  name: "upsample", type: "Deconvolution"
  bottom: "{{bottom_name}}" top: "{{top_name}}"
  convolution_param {
    kernel_size: {{2 * factor - factor % 2}} stride: {{factor}}
    num_output: {{C}} group: {{C}}
    pad: {{ceil((factor - 1) / 2.)}}
    weight_filler: { type: "bilinear" } bias_term: false
  }
  param { lr_mult: 0 decay_mult: 0 }
}
```

Please use this by replacing `{{}}` with your values. By specifying `num_output: {{C}} group: {{C}}`, it behaves as channel-wise convolution. The filter shape of this deconvolution layer will be (C, 1, K, K) where K is `kernel_size`, and this filler will set a (K, K) interpolation kernel for every channel of the filter identically. The resulting shape of the top feature map will be (B, C, factor $*$ H, factor $*$ W). Note that the learning rate and the weight decay are set to 0 in order to keep coefficient values of bilinear interpolation unchanged during training. If you apply this to an image, this operation is equivalent to the following call in Python with Scikit.Image.

```
out = skimage.transform.rescale(img, factor, mode='constant', cval=0)
```

The documentation for this class was generated from the following file:

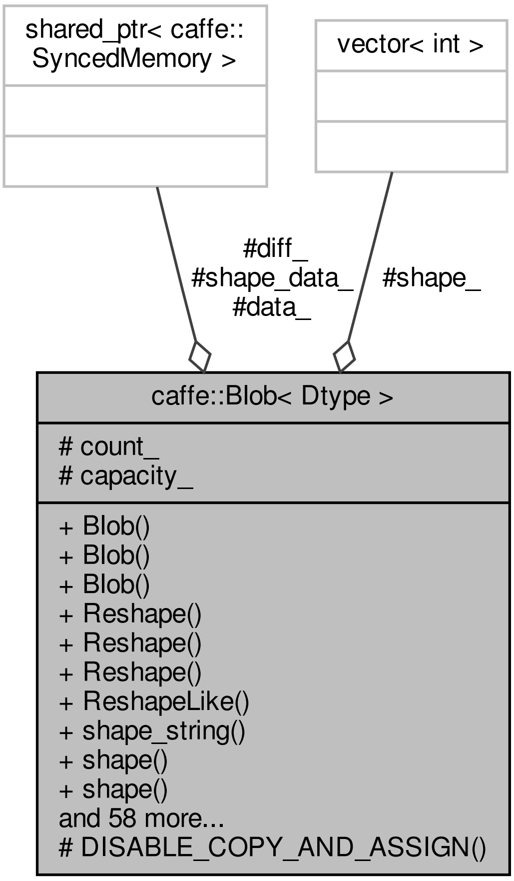
- include/caffe/filler.hpp

5.15 caffe::Blob< Dtype > Class Template Reference

A wrapper around SyncedMemory holders serving as the basic computational unit through which Layers, Nets, and Solvers interact.

`#include <blob.hpp>`

Collaboration diagram for caffe::Blob< Dtype >:



Public Member Functions

- Blob (const int num, const int channels, const int height, const int width)

    Deprecated; use `Blob(const vector<int>& shape)`.
- **Blob** (const vector< int > &shape)
- void Reshape (const int num, const int channels, const int height, const int width)

    Deprecated; use `Reshape(const vector<int>& shape)`.
- void Reshape (const vector< int > &shape)

    Change the dimensions of the blob, allocating new memory if necessary.
- void **Reshape** (const BlobShape &shape)
- void **ReshapeLike** (const Blob &other)
- string **shape_string** () const
- const vector< int > & **shape** () const
- int shape (int index) const

    Returns the dimension of the index-th axis (or the negative index-th axis from the end, if index is negative).
- int **num_axes** () const
- int **count** () const
- int count (int start_axis, int end_axis) const

    Compute the volume of a slice; i.e., the product of dimensions among a range of axes.
- int count (int start_axis) const

    Compute the volume of a slice spanning from a particular first axis to the final axis.
- int CanonicalAxisIndex (int axis_index) const

    Returns the 'canonical' version of a (usually) user-specified axis, allowing for negative indexing (e.g., -1 for the last axis).
- int num () const

    Deprecated legacy shape accessor num: use shape(0) instead.
- int channels () const

    Deprecated legacy shape accessor channels: use shape(1) instead.
- int height () const

    Deprecated legacy shape accessor height: use shape(2) instead.
- int width () const

    Deprecated legacy shape accessor width: use shape(3) instead.
- int **LegacyShape** (int index) const
- int **offset** (const int n, const int c=0, const int h=0, const int w=0) const
- int **offset** (const vector< int > &indices) const
- void CopyFrom (const Blob< Dtype > &source, bool copy_diff=false, bool reshape=false)

    Copy from a source Blob.
- Dtype **data_at** (const int n, const int c, const int h, const int w) const
- Dtype **diff_at** (const int n, const int c, const int h, const int w) const
- Dtype **data_at** (const vector< int > &index) const
- Dtype **diff_at** (const vector< int > &index) const
- const shared_ptr< SyncedMemory > & **data** () const
- const shared_ptr< SyncedMemory > & **diff** () const
- const Dtype * **cpu_data** () const
- void **set_cpu_data** (Dtype *data)
- const int * **gpu_shape** () const
- const Dtype * **gpu_data** () const
- void **set_gpu_data** (Dtype *data)
- const Dtype * **cpu_diff** () const
- const Dtype * **gpu_diff** () const
- Dtype * **mutable_cpu_data** ()
- Dtype * **mutable_gpu_data** ()
- Dtype * **mutable_cpu_diff** ()

- Dtype ∗ **mutable_gpu_diff** ()
- void **Update** ()
- void **FromProto** (const BlobProto &proto, bool reshape=true)
- void **ToProto** (BlobProto ∗proto, bool write_diff=false) const
- Dtype asum_data () const

  Compute the sum of absolute values (L1 norm) of the data.
- Dtype asum_diff () const

  Compute the sum of absolute values (L1 norm) of the diff.
- Dtype sumsq_data () const

  Compute the sum of squares (L2 norm squared) of the data.
- Dtype sumsq_diff () const

  Compute the sum of squares (L2 norm squared) of the diff.
- void scale_data (Dtype scale_factor)

  Scale the blob data by a constant factor.
- void scale_diff (Dtype scale_factor)

  Scale the blob diff by a constant factor.
- void ShareData (const Blob &other)

  Set the data_ shared_ptr to point to the SyncedMemory holding the data_ of Blob other – useful in Layers which simply perform a copy in their Forward pass.
- void ShareDiff (const Blob &other)

  Set the diff_ shared_ptr to point to the SyncedMemory holding the diff_ of Blob other – useful in Layers which simply perform a copy in their Forward pass.
- bool **ShapeEquals** (const BlobProto &other)
- template<>
  void **Update** ()
- template<>
  void **Update** ()
- template<>
  unsigned int **asum_data** () const
- template<>
  int **asum_data** () const
- template<>
  unsigned int **asum_diff** () const
- template<>
  int **asum_diff** () const
- template<>
  unsigned int **sumsq_data** () const
- template<>
  int **sumsq_data** () const
- template<>
  unsigned int **sumsq_diff** () const
- template<>
  int **sumsq_diff** () const
- template<>
  void **scale_data** (unsigned int scale_factor)
- template<>
  void **scale_data** (int scale_factor)
- template<>
  void **scale_diff** (unsigned int scale_factor)
- template<>
  void **scale_diff** (int scale_factor)
- template<>
  void **ToProto** (BlobProto ∗proto, bool write_diff) const
- template<>
  void **ToProto** (BlobProto ∗proto, bool write_diff) const

Protected Member Functions

- **DISABLE_COPY_AND_ASSIGN** (Blob)

Protected Attributes

- shared_ptr< SyncedMemory > **data_**
- shared_ptr< SyncedMemory > **diff_**
- shared_ptr< SyncedMemory > **shape_data_**
- vector< int > **shape_**
- int **count_**
- int **capacity_**

5.15.1 Detailed Description

template<typename Dtype>
class caffe::Blob< Dtype >

A wrapper around SyncedMemory holders serving as the basic computational unit through which Layers, Nets, and Solvers interact.

TODO(dox): more thorough description.

5.15.2 Member Function Documentation

5.15.2.1 CanonicalAxisIndex()

```
template<typename Dtype>
int caffe::Blob< Dtype >::CanonicalAxisIndex (
            int axis_index ) const  [inline]
```

Returns the 'canonical' version of a (usually) user-specified axis, allowing for negative indexing (e.g., -1 for the last axis).

Parameters

axis_index	the axis index. If 0 <= index < num_axes(), return index. If -num_axes <= index <= -1, return (num_axes() - (-index)), e.g., the last axis index (num_axes() - 1) if index == -1, the second to last if index == -2, etc. Dies on out of range index.

5.15.2.2 CopyFrom()

```
template<typename Dtype>
void caffe::Blob< Dtype >::CopyFrom (
```

```
              const Blob< Dtype > & source,
              bool copy_diff = false,
              bool reshape = false )
```

Copy from a source Blob.

Parameters

source	the Blob to copy from
copy_diff	if false, copy the data; if true, copy the diff
reshape	if false, require this Blob to be pre-shaped to the shape of other (and die otherwise); if true, Reshape this Blob to other's shape if necessary

5.15.2.3 count() [1/2]

```
template<typename Dtype>
int caffe::Blob< Dtype >::count (
              int start_axis,
              int end_axis ) const  [inline]
```

Compute the volume of a slice; i.e., the product of dimensions among a range of axes.

Parameters

start_axis	The first axis to include in the slice.
end_axis	The first axis to exclude from the slice.

5.15.2.4 count() [2/2]

```
template<typename Dtype>
int caffe::Blob< Dtype >::count (
              int start_axis ) const  [inline]
```

Compute the volume of a slice spanning from a particular first axis to the final axis.

Parameters

start_axis	The first axis to include in the slice.

5.15.2.5 Reshape()

```
template<typename Dtype >
void caffe::Blob< Dtype >::Reshape (
              const vector< int > & shape )
```

Change the dimensions of the blob, allocating new memory if necessary.

This function can be called both to create an initial allocation of memory, and to adjust the dimensions of a top blob during Layer::Reshape or Layer::Forward. When changing the size of blob, memory will only be reallocated if sufficient memory does not already exist, and excess memory will never be freed.

Note that reshaping an input blob and immediately calling Net::Backward is an error; either Net::Forward or Net::↩ Reshape need to be called to propagate the new input shape to higher layers.

5.15.2.6 shape()

```
template<typename Dtype>
int caffe::Blob< Dtype >::shape (
            int index ) const  [inline]
```

Returns the dimension of the index-th axis (or the negative index-th axis from the end, if index is negative).

Parameters

index	the axis index, which may be negative as it will be "canonicalized" using CanonicalAxisIndex. Dies on out of range index.

5.15.2.7 ShareData()

```
template<typename Dtype >
void caffe::Blob< Dtype >::ShareData (
            const Blob< Dtype > & other )
```

Set the data_ shared_ptr to point to the SyncedMemory holding the data_ of Blob other – useful in Layers which simply perform a copy in their Forward pass.

This deallocates the SyncedMemory holding this Blob's data_, as shared_ptr calls its destructor when reset with the "=" operator.

5.15.2.8 ShareDiff()

```
template<typename Dtype >
void caffe::Blob< Dtype >::ShareDiff (
            const Blob< Dtype > & other )
```

Set the diff_ shared_ptr to point to the SyncedMemory holding the diff_ of Blob other – useful in Layers which simply perform a copy in their Forward pass.

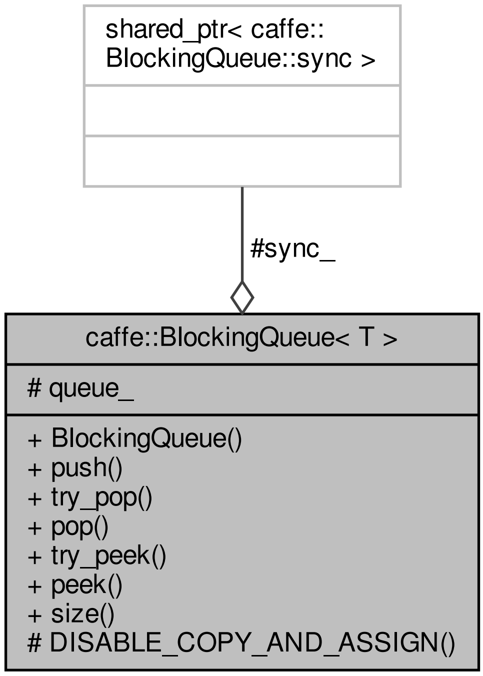
This deallocates the SyncedMemory holding this Blob's diff_, as shared_ptr calls its destructor when reset with the "=" operator.

The documentation for this class was generated from the following files:

- include/caffe/blob.hpp
- src/caffe/blob.cpp

5.16 caffe::BlockingQueue< T > Class Template Reference

Collaboration diagram for caffe::BlockingQueue< T >:



Classes

- class sync

Public Member Functions

- void **push** (const T &t)
- bool **try_pop** (T ∗t)
- T **pop** (const string &log_on_wait="")
- bool **try_peek** (T ∗t)
- T **peek** ()
- size_t **size** () const

Protected Member Functions

- **DISABLE_COPY_AND_ASSIGN** (BlockingQueue)

Protected Attributes

- std::queue< T > **queue_**
- shared_ptr< sync > **sync_**

The documentation for this class was generated from the following files:

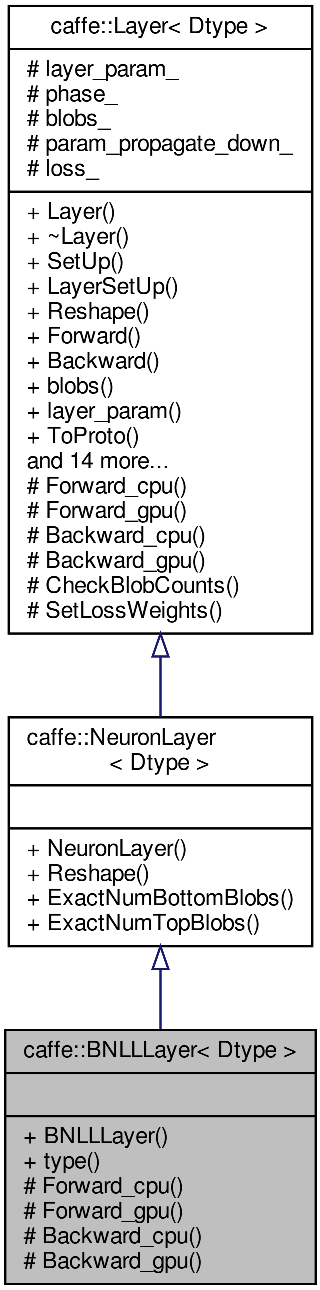
- include/caffe/util/blocking_queue.hpp
- src/caffe/util/blocking_queue.cpp

5.17 caffe::BNLLLayer< Dtype > Class Template Reference

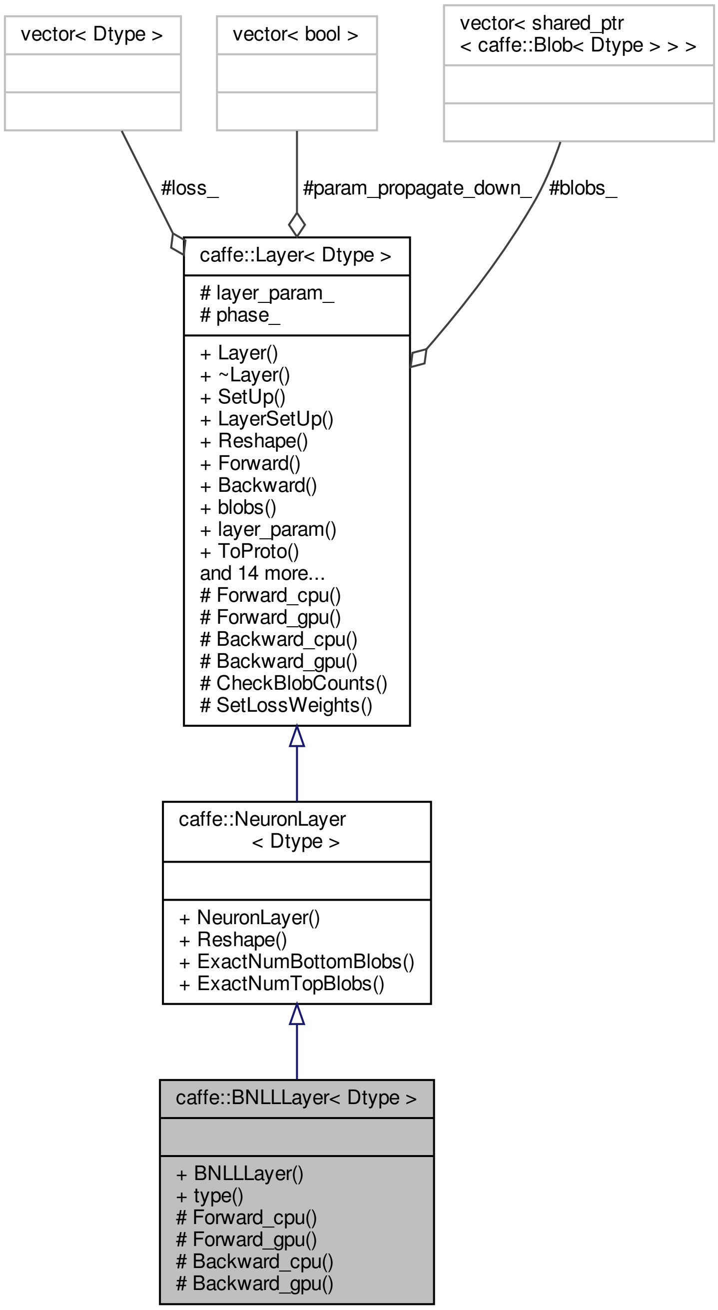
Computes $y = x + \log(1 + \exp(-x))$ if $x > 0$; $y = \log(1 + \exp(x))$ otherwise.

```
#include <bnll_layer.hpp>
```

Inheritance diagram for caffe::BNLLLayer< Dtype >:



Collaboration diagram for caffe::BNLLLayer< Dtype >:



Public Member Functions

- **BNLLLayer** (const LayerParameter &param)
- virtual const char ∗ type () const

  Returns the layer type.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Computes $y = x + \log(1 + \exp(-x))$ *if* $x > 0$*;* $y = \log(1 + \exp(x))$ *otherwise.*
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Computes the error gradient w.r.t. the BNLL inputs.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Additional Inherited Members

5.17.1 Detailed Description

template<typename Dtype>
class caffe::BNLLLayer< Dtype >

Computes $y = x + \log(1 + \exp(-x))$ if $x > 0$; $y = \log(1 + \exp(x))$ otherwise.

Parameters

bottom	input Blob vector (length 1) 1. $(N \times C \times H \times W)$ the inputs x
top	output Blob vector (length 1) 1. $(N \times C \times H \times W)$ the computed outputs $y = \begin{cases} x + \log(1 + \exp(-x)) & \text{if } x > 0 \\ \log(1 + \exp(x)) & \text{otherwise} \end{cases}$

5.17.2 Member Function Documentation

5.17.2.1 Backward_cpu()

```
template<typename Dtype >
void caffe::BNLLLayer< Dtype >::Backward_cpu (
            const vector< Blob< Dtype > *> & top,
            const vector< bool > & propagate_down,
            const vector< Blob< Dtype > *> & bottom )  [protected], [virtual]
```

Computes the error gradient w.r.t. the BNLL inputs.

Parameters

top	output Blob vector (length 1), providing the error gradient with respect to the outputs 1. $(N \times C \times H \times W)$ containing error gradients $\frac{\partial E}{\partial y}$ with respect to computed outputs y
propagate_down	see Layer::Backward.
bottom	input Blob vector (length 2) 1. $(N \times C \times H \times W)$ the inputs x; Backward fills their diff with gradients $\frac{\partial E}{\partial x}$ if propagate_down[0]

Implements caffe::Layer< Dtype >.

5.17.2.2 Forward_cpu()

```
template<typename Dtype >
void caffe::BNLLLayer< Dtype >::Forward_cpu (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [protected], [virtual]
```

Computes $y = x + \log(1 + \exp(-x))$ if $x > 0$; $y = \log(1 + \exp(x))$ otherwise.

Parameters

bottom	input Blob vector (length 1) 1. $(N \times C \times H \times W)$ the inputs x
top	output Blob vector (length 1) 1. $(N \times C \times H \times W)$ the computed outputs $y = \begin{cases} x + \log(1 + \exp(-x)) & \text{if } x > 0 \\ \log(1 + \exp(x)) & \text{otherwise} \end{cases}$

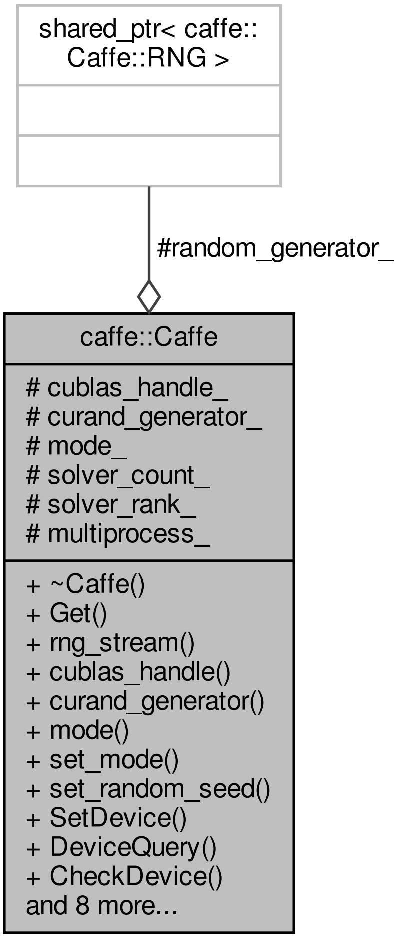
Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

- include/caffe/layers/bnll_layer.hpp
- src/caffe/layers/bnll_layer.cpp

5.18 caffe::Caffe Class Reference

Collaboration diagram for caffe::Caffe:



Classes

- class RNG

Public Types

- enum **Brew** { **CPU**, **GPU** }

Static Public Member Functions

- static Caffe & **Get** ()
- static RNG & **rng_stream** ()
- static cublasHandle_t **cublas_handle** ()
- static curandGenerator_t **curand_generator** ()
- static Brew **mode** ()
- static void **set_mode** (Brew mode)
- static void **set_random_seed** (const unsigned int seed)
- static void **SetDevice** (const int device_id)
- static void **DeviceQuery** ()
- static bool **CheckDevice** (const int device_id)
- static int **FindDevice** (const int start_id=0)
- static int **solver_count** ()
- static void **set_solver_count** (int val)
- static int **solver_rank** ()
- static void **set_solver_rank** (int val)
- static bool **multiprocess** ()
- static void **set_multiprocess** (bool val)
- static bool **root_solver** ()

Protected Attributes

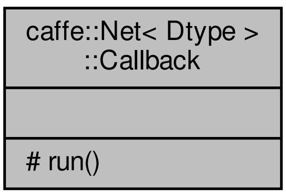
- cublasHandle_t **cublas_handle_**
- curandGenerator_t **curand_generator_**
- shared_ptr< RNG > **random_generator_**
- Brew **mode_**
- int **solver_count_**
- int **solver_rank_**
- bool **multiprocess_**

The documentation for this class was generated from the following files:

- include/caffe/common.hpp
- src/caffe/common.cpp

5.19 caffe::Net< Dtype >::Callback Class Reference

Collaboration diagram for caffe::Net< Dtype >::Callback:



Protected Member Functions

- virtual void **run** (int layer)=0

Friends

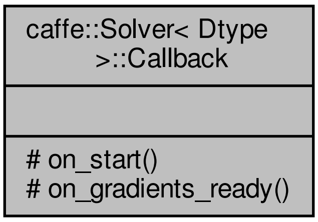
- template<typename T >
  class **Net**

The documentation for this class was generated from the following file:

- include/caffe/net.hpp

5.20 caffe::Solver< Dtype >::Callback Class Reference

Collaboration diagram for caffe::Solver< Dtype >::Callback:



Protected Member Functions

- virtual void **on_start** ()=0
- virtual void **on_gradients_ready** ()=0

Friends

- template<typename T >
  class **Solver**

The documentation for this class was generated from the following file:

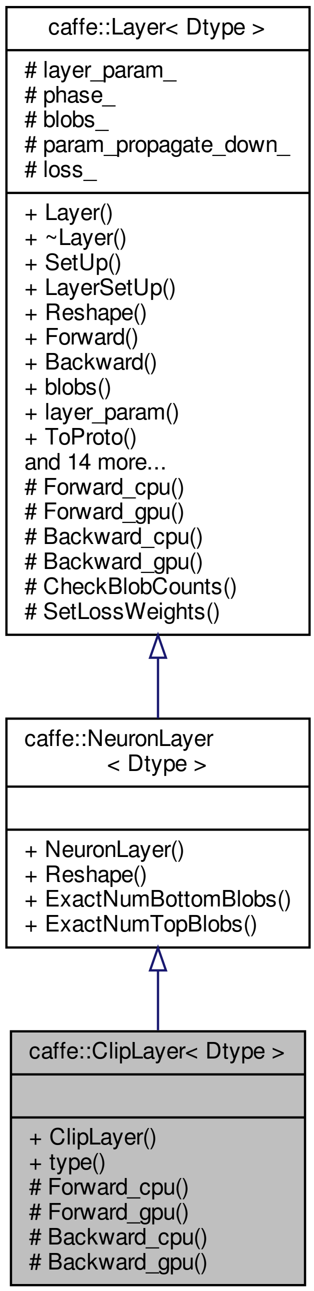
- include/caffe/solver.hpp

5.21 caffe::ClipLayer< Dtype > Class Template Reference

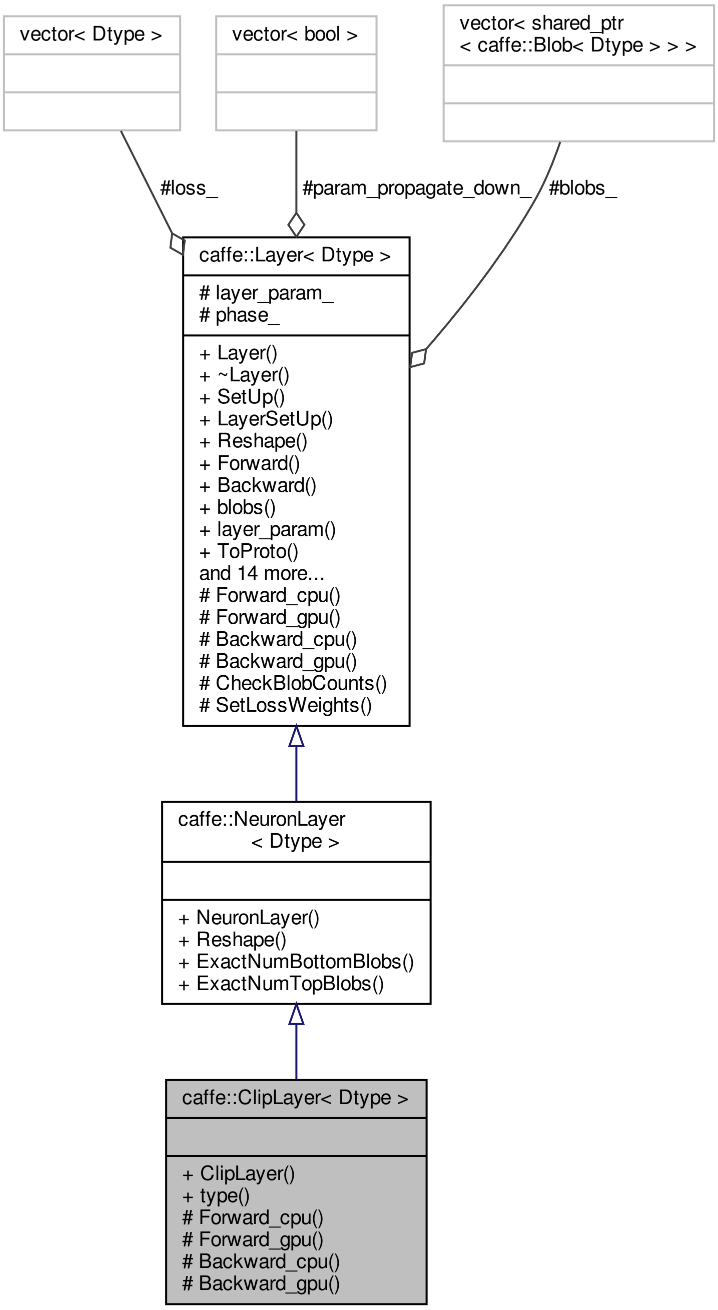
Clip: $y = \max(min, \min(max, x))$.

```
#include <clip_layer.hpp>
```

Inheritance diagram for caffe::ClipLayer< Dtype >:



Collaboration diagram for caffe::ClipLayer< Dtype >:



Public Member Functions

- ClipLayer (const LayerParameter &param)
- virtual const char ∗ type () const

    Returns the layer type.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

    Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

    Computes the error gradient w.r.t. the clipped inputs.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

    Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Additional Inherited Members

5.21.1 Detailed Description

template<typename Dtype>
class caffe::ClipLayer< Dtype >

Clip: $y = \max(min, \min(max, x))$.

5.21.2 Constructor & Destructor Documentation

5.21.2.1 ClipLayer()

```
template<typename Dtype >
caffe::ClipLayer< Dtype >::ClipLayer (
            const LayerParameter & param )  [inline], [explicit]
```

Parameters

param	provides ClipParameter clip_param, with ClipLayer options: • min • max

5.21.3 Member Function Documentation

5.21.3.1 Backward_cpu()

```
template<typename Dtype >
void caffe::ClipLayer< Dtype >::Backward_cpu (
            const vector< Blob< Dtype > *> & top,
            const vector< bool > & propagate_down,
            const vector< Blob< Dtype > *> & bottom )  [protected], [virtual]
```

Computes the error gradient w.r.t. the clipped inputs.

Parameters

top	output Blob vector (length 1), providing the error gradient with respect to the outputs 1. $(N \times C \times H \times W)$ containing error gradients $\frac{\partial E}{\partial y}$ with respect to computed outputs y
propagate_down	see Layer::Backward.
bottom	input Blob vector (length 1) 1. $(N \times C \times H \times W)$ the inputs x; Backward fills their diff with gradients $\frac{\partial E}{\partial x} = \left\{ \begin{array}{lr} 0 & \mathrm{if} \; x < min \vee x > max \\ \frac{\partial E}{\partial y} & \mathrm{if} \; x \ge min \wedge x \le max \end{array} \right.$

Implements caffe::Layer< Dtype >.

5.21.3.2 Forward_cpu()

```
template<typename Dtype >
void caffe::ClipLayer< Dtype >::Forward_cpu (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [protected], [virtual]
```

Parameters

bottom	input Blob vector (length 1) 1. $(N \times C \times H \times W)$ the inputs x
top	output Blob vector (length 1) 1. $(N \times C \times H \times W)$ the computed outputs $y = \max(min, \min(max, x))$

Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

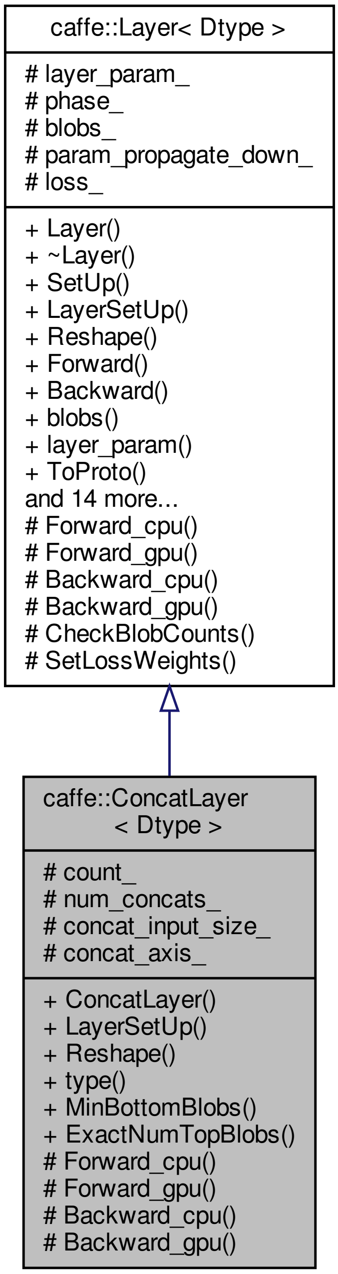
- include/caffe/layers/clip_layer.hpp
- src/caffe/layers/clip_layer.cpp

5.22 caffe::ConcatLayer< Dtype > Class Template Reference

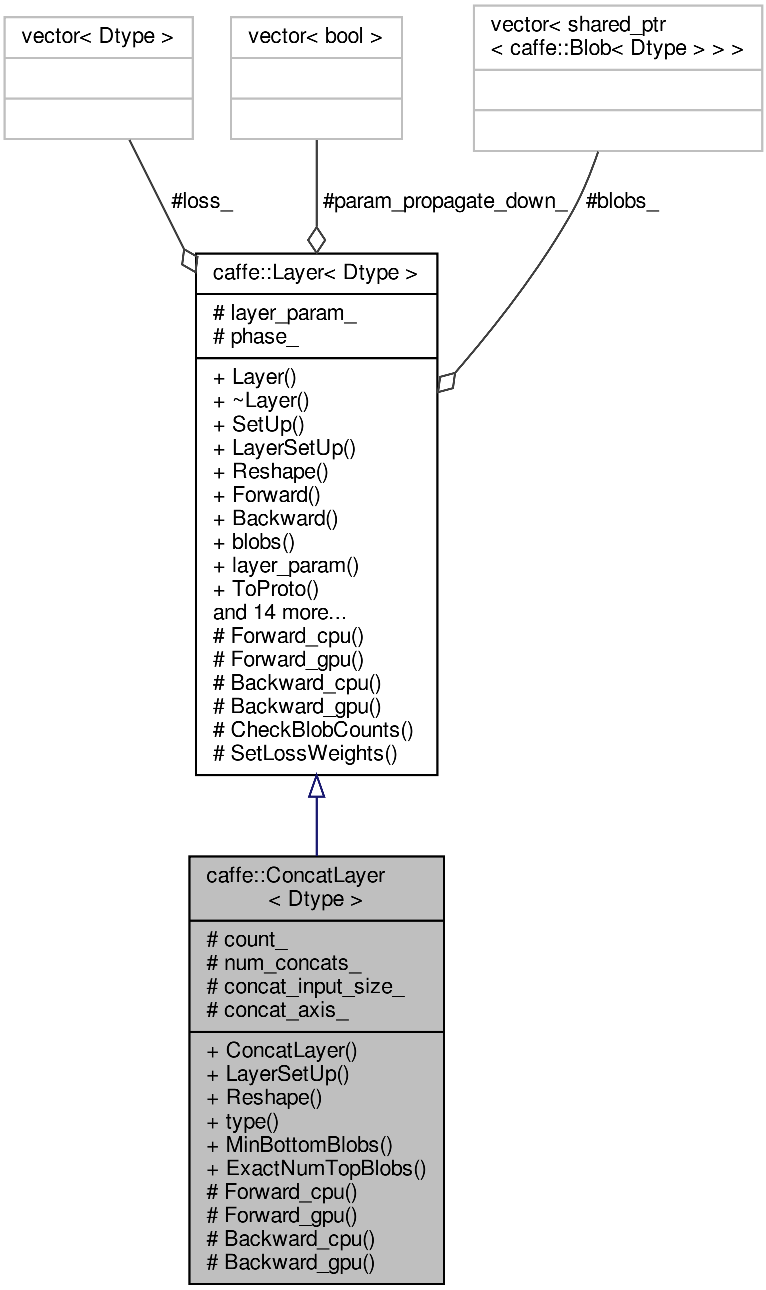
Takes at least two Blobs and concatenates them along either the num or channel dimension, outputting the result.

```
#include <concat_layer.hpp>
```

Inheritance diagram for caffe::ConcatLayer< Dtype >:



Collaboration diagram for caffe::ConcatLayer< Dtype >:



Public Member Functions

- **ConcatLayer** (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

    Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void Reshape (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

    Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

- virtual const char ∗ type () const

  Returns the layer type.
- virtual int MinBottomBlobs () const

  Returns the minimum number of bottom blobs required by the layer, or -1 if no minimum number is required.
- virtual int ExactNumTopBlobs () const

  Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)
- virtual void Forward_gpu (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > ∗> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > ∗> &bottom)

  Computes the error gradient w.r.t. the concatenate inputs.
- virtual void Backward_gpu (const vector< Blob< Dtype > ∗> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > ∗> &bottom)

  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Protected Attributes

- int **count_**
- int **num_concats_**
- int **concat_input_size_**
- int **concat_axis_**

5.22.1 Detailed Description

template<typename Dtype>
class caffe::ConcatLayer< Dtype >

Takes at least two Blobs and concatenates them along either the num or channel dimension, outputting the result.

5.22.2 Member Function Documentation

5.22.2.1 Backward_cpu()

```
template<typename Dtype >
void caffe::ConcatLayer< Dtype >::Backward_cpu (
            const vector< Blob< Dtype > *> & top,
            const vector< bool > & propagate_down,
            const vector< Blob< Dtype > *> & bottom ) [protected], [virtual]
```

Computes the error gradient w.r.t. the concatenate inputs.

Parameters

top	output Blob vector (length 1), providing the error gradient with respect to the outputs 1. $(KN \times C \times H \times W)$ if axis == 0, or $(N \times KC \times H \times W)$ if axis == 1: containing error gradients $\frac{\partial E}{\partial y}$ with respect to concatenated outputs y
propagate_down	see Layer::Backward.
bottom	input Blob vector (length K), into which the top gradient $\frac{\partial E}{\partial y}$ is deconcatenated back to the inputs $\left[\begin{array}{cccc} \frac{\partial E}{\partial x_1} & \frac{\partial E}{\partial x_2} & \dots & \frac{\partial E}{\partial x_K} \end{array} \right] = \frac{\partial E}{\partial y}$

Implements caffe::Layer< Dtype >.

5.22.2.2 ExactNumTopBlobs()

```
template<typename Dtype >
virtual int caffe::ConcatLayer< Dtype >::ExactNumTopBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.22.2.3 Forward_cpu()

```
template<typename Dtype >
void caffe::ConcatLayer< Dtype >::Forward_cpu (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [protected], [virtual]
```

Parameters

bottom	input Blob vector (length 2+) 1. $(N \times C \times H \times W)$ the inputs x_1 2. $(N \times C \times H \times W)$ the inputs x_2 3. ... • K $(N \times C \times H \times W)$ the inputs x_K
top	output Blob vector (length 1) 1. $(KN \times C \times H \times W)$ if axis == 0, or $(N \times KC \times H \times W)$ if axis == 1: the concatenated output $y = \left[\begin{array}{cccc} x_1 & x_2 & \dots & x_K \end{array} \right]$

Implements caffe::Layer< Dtype >.

5.22.2.4 LayerSetUp()

```
template<typename Dtype >
void caffe::ConcatLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

5.22.2.5 MinBottomBlobs()

```
template<typename Dtype >
virtual int caffe::ConcatLayer< Dtype >::MinBottomBlobs ( ) const  [inline], [virtual]
```

Returns the minimum number of bottom blobs required by the layer, or -1 if no minimum number is required.

This method should be overridden to return a non-negative value if your layer expects some minimum number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.22.2.6 Reshape()

```
template<typename Dtype >
void caffe::ConcatLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Implements caffe::Layer$<$ Dtype $>$.

The documentation for this class was generated from the following files:

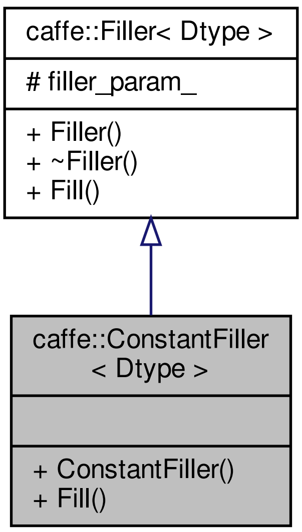
- include/caffe/layers/concat_layer.hpp
- src/caffe/layers/concat_layer.cpp

5.23 caffe::ConstantFiller$<$ Dtype $>$ Class Template Reference

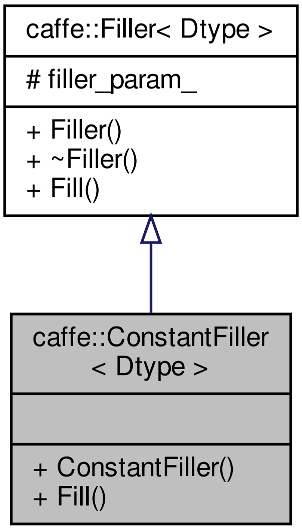
Fills a Blob with constant values $x = 0$.

`#include <filler.hpp>`

Inheritance diagram for caffe::ConstantFiller$<$ Dtype $>$:



Collaboration diagram for caffe::ConstantFiller< Dtype >:



Public Member Functions

- **ConstantFiller** (const FillerParameter &param)
- virtual void **Fill** (Blob< Dtype > *blob)

Additional Inherited Members

5.23.1 Detailed Description

template<typename Dtype>
class caffe::ConstantFiller< Dtype >

Fills a Blob with constant values $x = 0$.

The documentation for this class was generated from the following file:

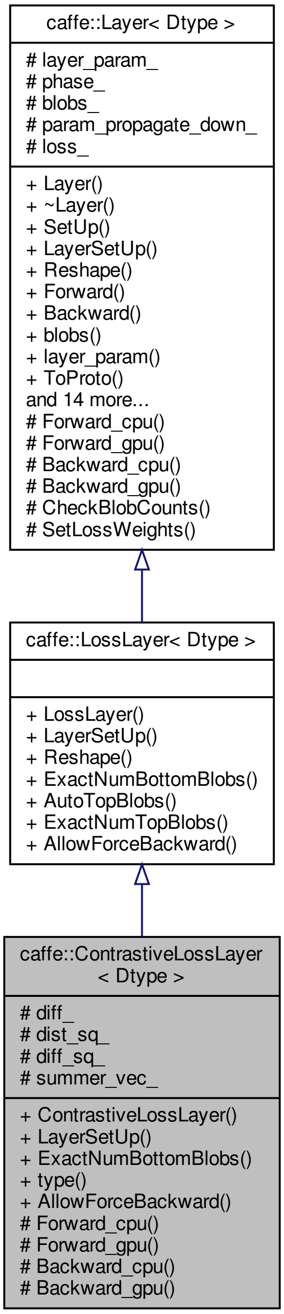
- include/caffe/filler.hpp

5.24 caffe::ContrastiveLossLayer< Dtype > Class Template Reference

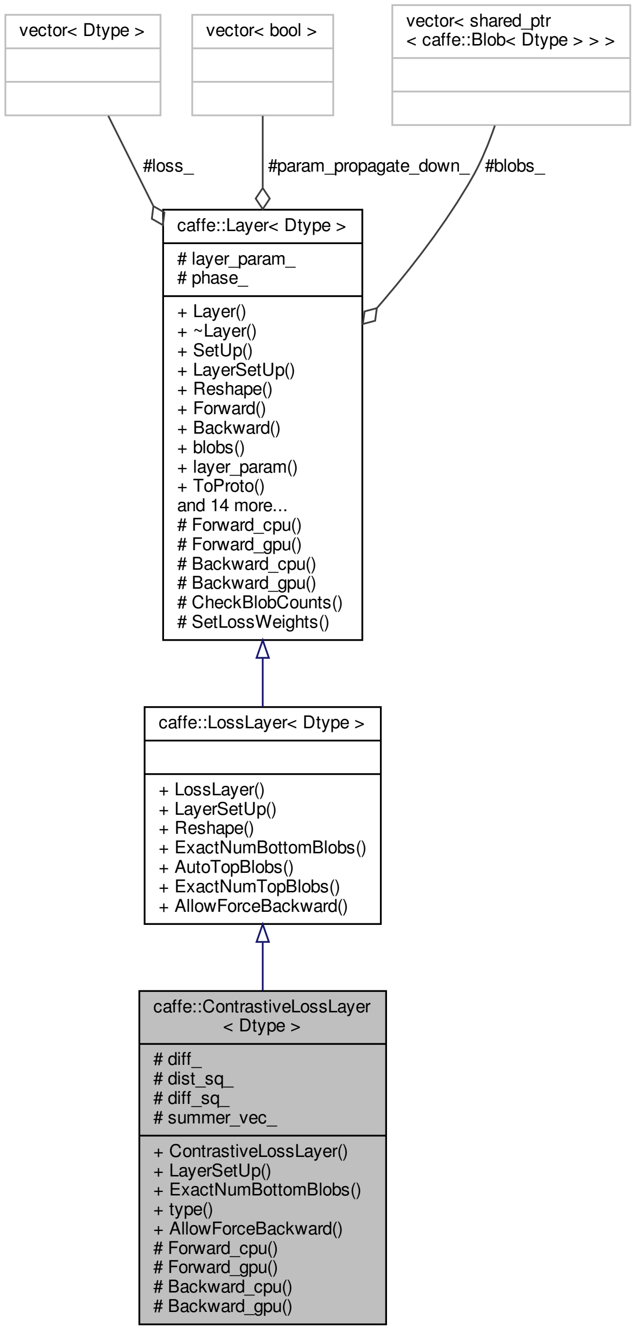
Computes the contrastive loss $E = \frac{1}{2N} \sum\limits_{n=1}^{N} \left(y\right) d^2 + \left(1-y\right) \max \left(margin-d, 0\right)^2$ where $d = \left| \left| a_n - b_n \right| \right|_2$. This can be used to train siamese networks.

```
#include <contrastive_loss_layer.hpp>
```

Inheritance diagram for caffe::ContrastiveLossLayer< Dtype >:



Collaboration diagram for caffe::ContrastiveLossLayer< Dtype >:



Public Member Functions

- **ContrastiveLossLayer** (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual int ExactNumBottomBlobs () const

  Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

- virtual const char $*$ type () const

    Returns the layer type.
- virtual bool AllowForceBackward (const int bottom_index) const

Protected Member Functions

- virtual void Forward_cpu (const vector$<$ Blob$<$ Dtype $>$ $*>$ &bottom, const vector$<$ Blob$<$ Dtype $>$ $*>$ &top)

    Computes the contrastive loss $E = \frac{1}{2N} \sum\limits_{n=1}^{N} (y)\, d^2 + (1-y) \max\left(margin - d, 0\right)^2$ *where* $d = \left\| a_n - b_n \right\|_2$. *This can be used to train siamese networks.*
- virtual void Forward_gpu (const vector$<$ Blob$<$ Dtype $>$ $*>$ &bottom, const vector$<$ Blob$<$ Dtype $>$ $*>$ &top)

    Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector$<$ Blob$<$ Dtype $>$ $*>$ &top, const vector$<$ bool $>$ &propagate_down, const vector$<$ Blob$<$ Dtype $>$ $*>$ &bottom)

    Computes the Contrastive error gradient w.r.t. the inputs.
- virtual void Backward_gpu (const vector$<$ Blob$<$ Dtype $>$ $*>$ &top, const vector$<$ bool $>$ &propagate_down, const vector$<$ Blob$<$ Dtype $>$ $*>$ &bottom)

    Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Protected Attributes

- Blob$<$ Dtype $>$ **diff_**
- Blob$<$ Dtype $>$ **dist_sq_**
- Blob$<$ Dtype $>$ **diff_sq_**
- Blob$<$ Dtype $>$ **summer_vec_**

5.24.1 Detailed Description

template$<$typename Dtype$>$
class caffe::ContrastiveLossLayer$<$ Dtype $>$

Computes the contrastive loss $E = \frac{1}{2N} \sum\limits_{n=1}^{N} (y)\, d^2 + (1-y) \max\left(margin - d, 0\right)^2$ where $d = \left\| a_n - b_n \right\|_2$. This can be used to train siamese networks.

Parameters

bottom	input Blob vector (length 3) 1. $(N \times C \times 1 \times 1)$ the features $a \in [-\infty, +\infty]$ 2. $(N \times C \times 1 \times 1)$ the features $b \in [-\infty, +\infty]$ 3. $(N \times 1 \times 1 \times 1)$ the binary similarity $s \in [0, 1]$
top	output Blob vector (length 1) 1. $(1 \times 1 \times 1 \times 1)$ the computed contrastive loss: $E = \frac{1}{2N} \sum\limits_{n=1}^{N} (y)\, d^2 + (1-y) \max\left(margin - d, 0\right)^2$ where $d = \left\| a_n - b_n \right\|_2$. This can be used to train siamese networks.

5.24.2 Member Function Documentation

5.24.2.1 AllowForceBackward()

```
template<typename Dtype >
virtual bool caffe::ContrastiveLossLayer< Dtype >::AllowForceBackward (
            const int bottom_index ) const  [inline], [virtual]
```

Unlike most loss layers, in the ContrastiveLossLayer we can backpropagate to the first two inputs.

Reimplemented from caffe::LossLayer< Dtype >.

5.24.2.2 Backward_cpu()

```
template<typename Dtype >
void caffe::ContrastiveLossLayer< Dtype >::Backward_cpu (
            const vector< Blob< Dtype > *> & top,
            const vector< bool > & propagate_down,
            const vector< Blob< Dtype > *> & bottom )  [protected], [virtual]
```

Computes the Contrastive error gradient w.r.t. the inputs.

Computes the gradients with respect to the two input vectors (bottom[0] and bottom[1]), but not the similarity label (bottom[2]).

Parameters

top	output Blob vector (length 1), providing the error gradient with respect to the outputs 1. $(1 \times 1 \times 1 \times 1)$ This Blob's diff will simply contain the loss_weight$*$ λ, as λ is the coefficient of this layer's output ℓ_i in the overall Net loss $E = \lambda_i \ell_i +$ other loss terms; hence $\frac{\partial E}{\partial \ell_i} = \lambda_i$. ($*$Assuming that this top Blob is not used as a bottom (input) by any other layer of the Net.)
propagate_down	see Layer::Backward.
bottom	input Blob vector (length 2) 1. $(N \times C \times 1 \times 1)$ the features a; Backward fills their diff with gradients if propagate_down[0] 2. $(N \times C \times 1 \times 1)$ the features b; Backward fills their diff with gradients if propagate_down[1]

Implements caffe::Layer< Dtype >.

5.24.2.3 ExactNumBottomBlobs()

```
template<typename Dtype >
virtual int caffe::ContrastiveLossLayer< Dtype >::ExactNumBottomBlobs ( ) const  [inline],
[virtual]
```

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of bottom blobs.

Reimplemented from caffe::LossLayer< Dtype >.

5.24.2.4 Forward_cpu()

```
template<typename Dtype >
void caffe::ContrastiveLossLayer< Dtype >::Forward_cpu (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [protected], [virtual]
```

Computes the contrastive loss $E = \frac{1}{2N} \sum\limits_{n=1}^{N} \left(y\right) d^2 + \left(1-y\right) \max \left(margin-d, 0\right)^2$ where $d = \left| \left| a_n - b_n \right| \right|_2$. This can be used to train siamese networks.

Parameters

bottom	input Blob vector (length 3) 1. $(N \times C \times 1 \times 1)$ the features $a \in [-\infty, +\infty]$ 2. $(N \times C \times 1 \times 1)$ the features $b \in [-\infty, +\infty]$ 3. $(N \times 1 \times 1 \times 1)$ the binary similarity $s \in [0, 1]$
top	output Blob vector (length 1) 1. $(1 \times 1 \times 1 \times 1)$ the computed contrastive loss: $E = \frac{1}{2N} \sum\limits_{n=1}^{N} \left(y\right) d^2 + \left(1-y\right) \max \left(margin-d, 0\right)^2$ where $d = \left| \left| a_n - b_n \right| \right|_2$. This can be used to train siamese networks.

Implements caffe::Layer< Dtype >.

5.24.2.5 LayerSetUp()

```
template<typename Dtype >
void caffe::ContrastiveLossLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::LossLayer< Dtype >.

The documentation for this class was generated from the following files:

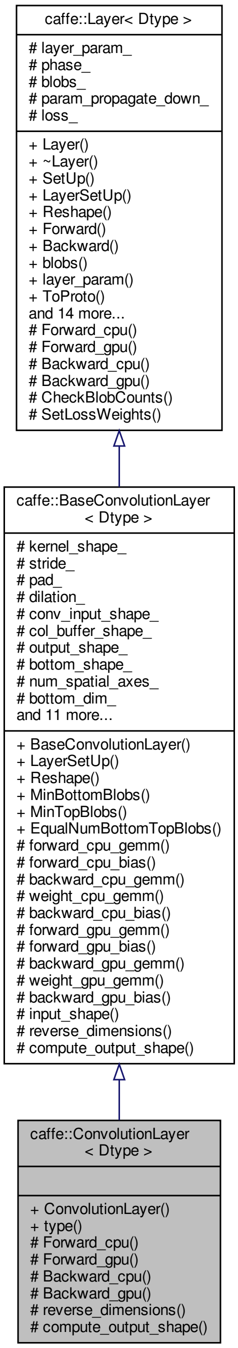
- include/caffe/layers/contrastive_loss_layer.hpp
- src/caffe/layers/contrastive_loss_layer.cpp

5.25 caffe::ConvolutionLayer< Dtype > Class Template Reference

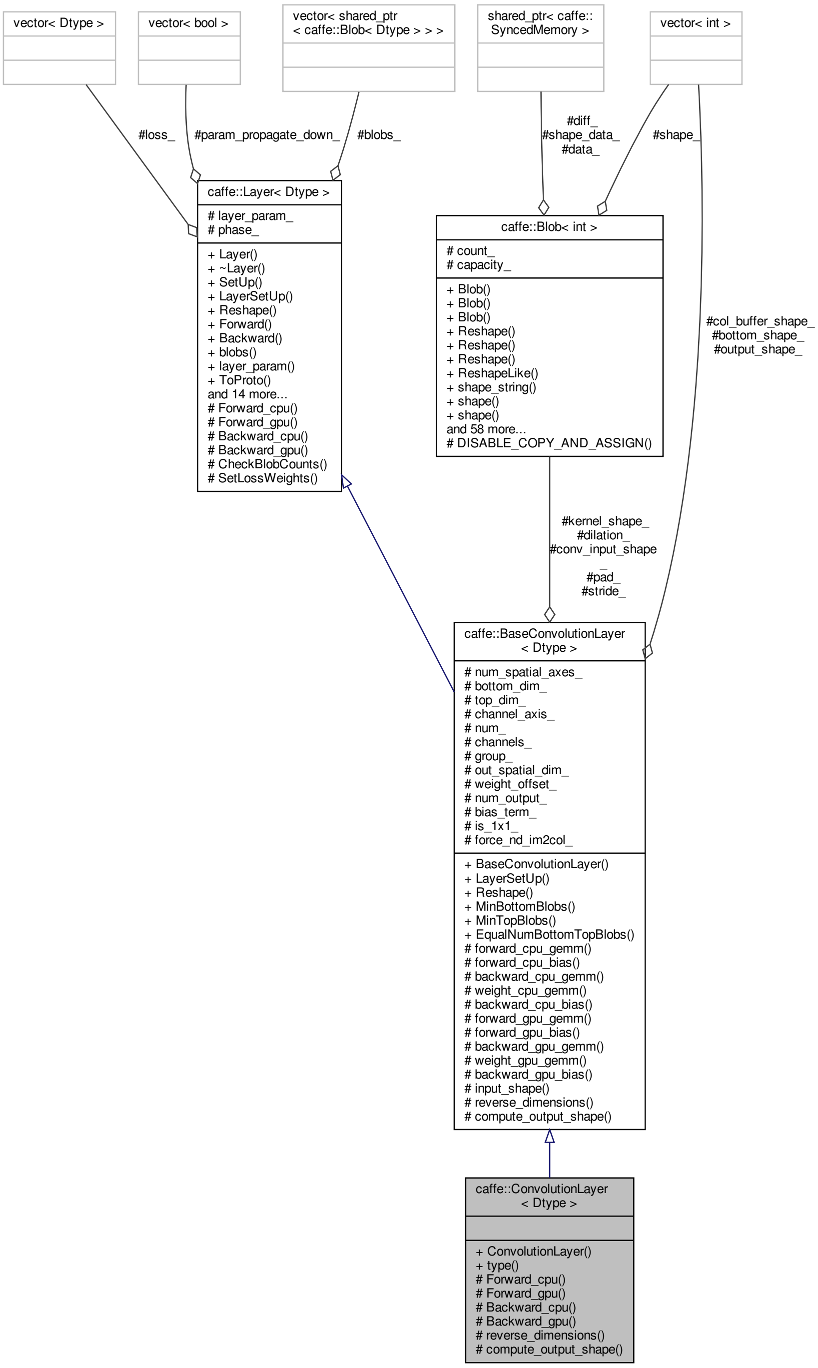
Convolves the input image with a bank of learned filters, and (optionally) adds biases.

`#include <conv_layer.hpp>`

Inheritance diagram for caffe::ConvolutionLayer< Dtype >:



Collaboration diagram for caffe::ConvolutionLayer< Dtype >:



Public Member Functions

- ConvolutionLayer (const LayerParameter &param)
- virtual const char ∗ type () const

  Returns the layer type.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

    Using the CPU device, compute the layer output.
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

    Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

    Using the CPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

    Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.
- virtual bool **reverse_dimensions** ()
- virtual void **compute_output_shape** ()

Additional Inherited Members

5.25.1 Detailed Description

template<typename Dtype>
class caffe::ConvolutionLayer< Dtype >

Convolves the input image with a bank of learned filters, and (optionally) adds biases.

Caffe convolves by reduction to matrix multiplication. This achieves high-throughput and generality of input and filter dimensions but comes at the cost of memory for matrices. This makes use of efficiency in BLAS.

The input is "im2col" transformed to a channel K' x H x W data matrix for multiplication with the N x K' x H x W filter matrix to yield a N' x H x W output matrix that is then "col2im" restored. K' is the input channel * kernel height * kernel width dimension of the unrolled inputs so that the im2col matrix has a column for each input region to be filtered. col2im restores the output spatial structure by rolling up the output channel N' columns of the output matrix.

5.25.2 Constructor & Destructor Documentation

5.25.2.1 ConvolutionLayer()

```
template<typename Dtype >
caffe::ConvolutionLayer< Dtype >::ConvolutionLayer (
            const LayerParameter & param )  [inline], [explicit]
```

Parameters

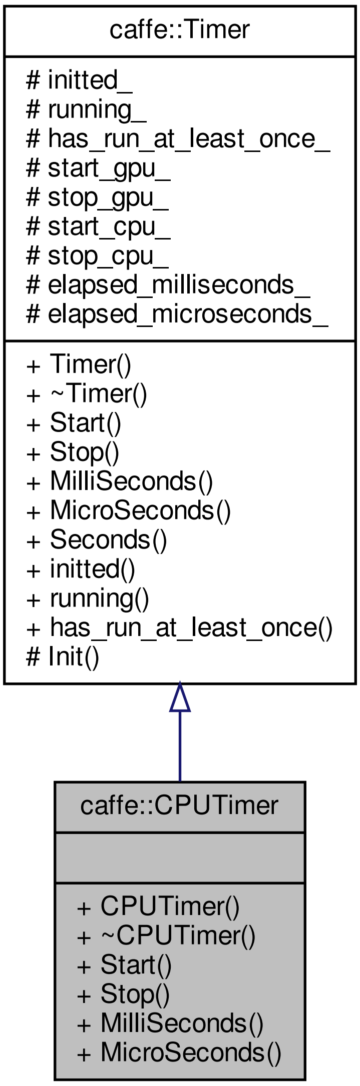
param	provides ConvolutionParameter convolution_param, with ConvolutionLayer options: • num_output. The number of filters. • kernel_size / kernel_h / kernel_w. The filter dimensions, given by kernel_size for square filters or kernel_h and kernel_w for rectangular filters. • stride / stride_h / stride_w (**optional**, default 1). The filter stride, given by stride_size for equal dimensions or stride_h and stride_w for different strides. By default the convolution is dense with stride 1. • pad / pad_h / pad_w (**optional**, default 0). The zero-padding for convolution, given by pad for equal dimensions or pad_h and pad_w for different padding. Input padding is computed implicitly instead of actually padding. • dilation (**optional**, default 1). The filter dilation, given by dilation_size for equal dimensions for different dilation. By default the convolution has dilation 1. • group (**optional**, default 1). The number of filter groups. Group convolution is a method for reducing parameterization by selectively connecting input and output channels. The input and output channel dimensions must be divisible by the number of groups. For group ≥ 1, the convolutional filters' input and output channels are separated s.t. each group takes 1 / group of the input channels and makes 1 / group of the output channels. Concretely 4 input channels, 8 output channels, and 2 groups separate input channels 1-2 and output channels 1-4 into the first group and input channels 3-4 and output channels 5-8 into the second group. • bias_term (**optional**, default true). Whether to have a bias. • engine: convolution has CAFFE (matrix multiplication) and CUDNN (library kernels + stream parallelism) engines.

The documentation for this class was generated from the following files:

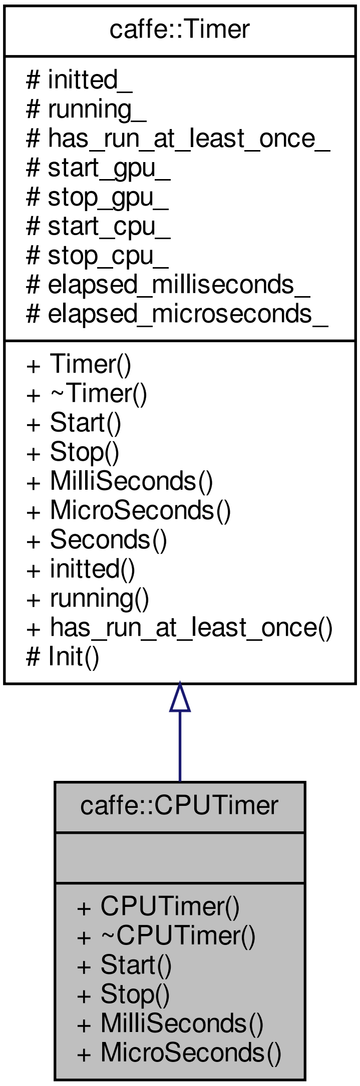
- include/caffe/layers/conv_layer.hpp
- src/caffe/layers/conv_layer.cpp

5.26 caffe::CPUTimer Class Reference

Inheritance diagram for caffe::CPUTimer:



Collaboration diagram for caffe::CPUTimer:



Public Member Functions

- virtual void **Start** ()
- virtual void **Stop** ()
- virtual float **MilliSeconds** ()
- virtual float **MicroSeconds** ()

Additional Inherited Members

The documentation for this class was generated from the following files:

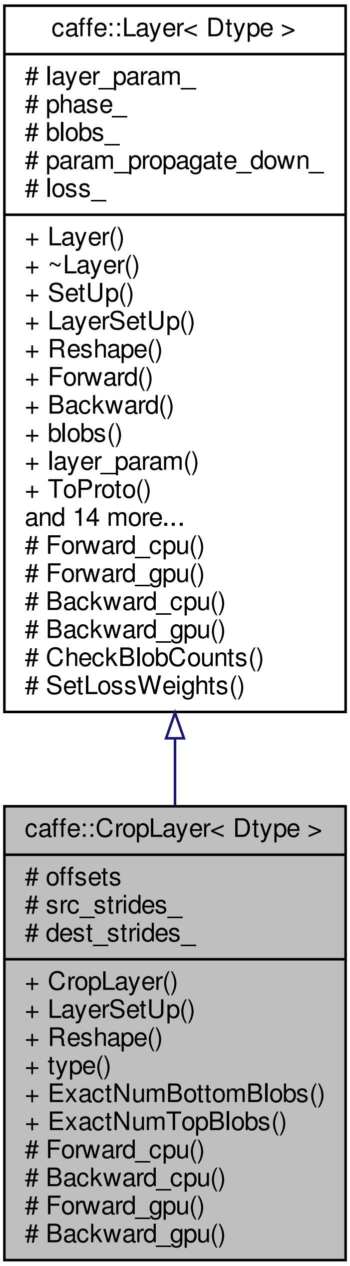
- include/caffe/util/benchmark.hpp
- src/caffe/util/benchmark.cpp

5.27 caffe::CropLayer< Dtype > Class Template Reference

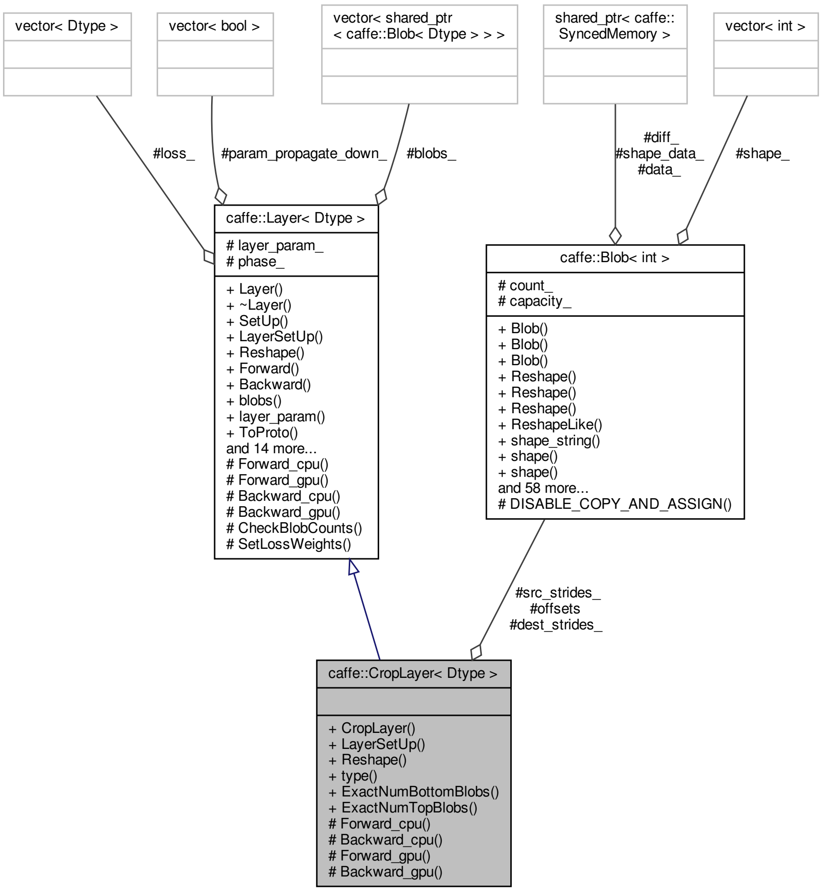
Takes a Blob and crop it, to the shape specified by the second input Blob, across all dimensions after the specified axis.

```
#include <crop_layer.hpp>
```

Inheritance diagram for caffe::CropLayer< Dtype >:



Collaboration diagram for caffe::CropLayer< Dtype >:



Public Member Functions

- **CropLayer** (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

    Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void Reshape (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

    Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.
- virtual const char ∗ type () const

    Returns the layer type.
- virtual int ExactNumBottomBlobs () const

    Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.
- virtual int ExactNumTopBlobs () const

    Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

Using the CPU device, compute the layer output.

- virtual void Backward_cpu (const vector< Blob< Dtype > ∗> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > ∗> &bottom)

  Using the CPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true.

- virtual void Forward_gpu (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.

- virtual void Backward_gpu (const vector< Blob< Dtype > ∗> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > ∗> &bottom)

  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Protected Attributes

- Blob< int > **offsets**
- Blob< int > **src_strides_**
- Blob< int > **dest_strides_**

5.27.1 Detailed Description

template<typename Dtype>
class caffe::CropLayer< Dtype >

Takes a Blob and crop it, to the shape specified by the second input Blob, across all dimensions after the specified axis.

TODO(dox): thorough documentation for Forward, Backward, and proto params.

5.27.2 Member Function Documentation

5.27.2.1 ExactNumBottomBlobs()

```
template<typename Dtype >
virtual int caffe::CropLayer< Dtype >::ExactNumBottomBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.27.2.2 ExactNumTopBlobs()

```
template<typename Dtype >
virtual int caffe::CropLayer< Dtype >::ExactNumTopBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.27.2.3 LayerSetUp()

```
template<typename Dtype >
void caffe::CropLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

5.27.2.4 Reshape()

```
template<typename Dtype >
void caffe::CropLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as

reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

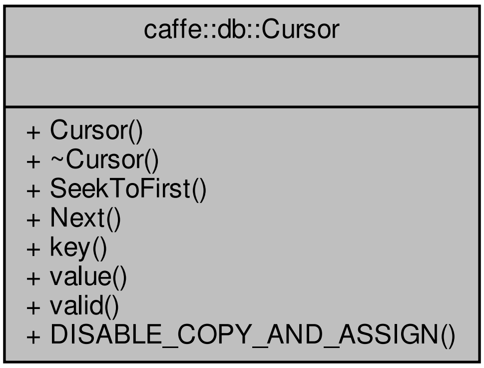
Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

- include/caffe/layers/crop_layer.hpp
- src/caffe/layers/crop_layer.cpp

5.28 caffe::db::Cursor Class Reference

Collaboration diagram for caffe::db::Cursor:



Public Member Functions

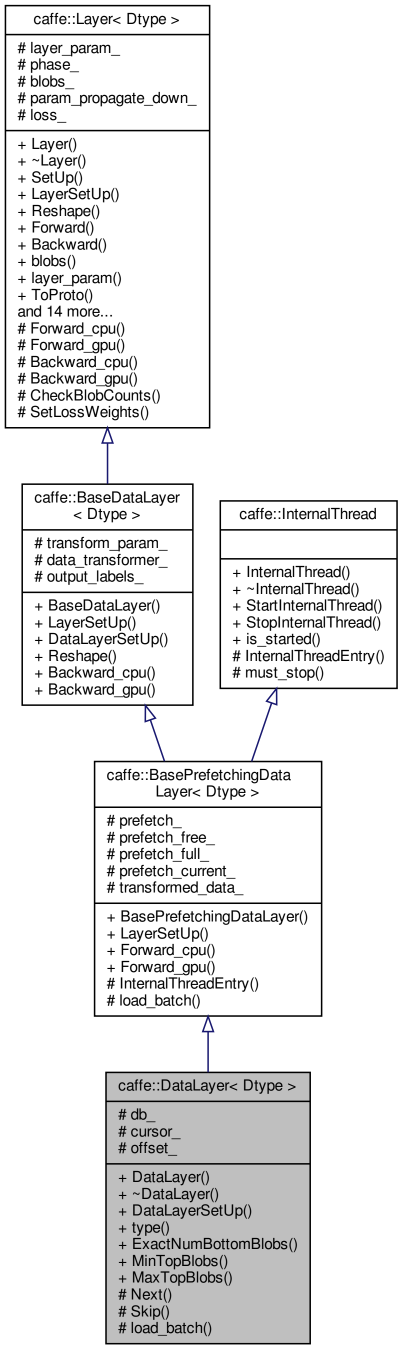
- virtual void **SeekToFirst** ()=0
- virtual void **Next** ()=0
- virtual string **key** ()=0
- virtual string **value** ()=0
- virtual bool **valid** ()=0
- **DISABLE_COPY_AND_ASSIGN** (Cursor)

The documentation for this class was generated from the following file:

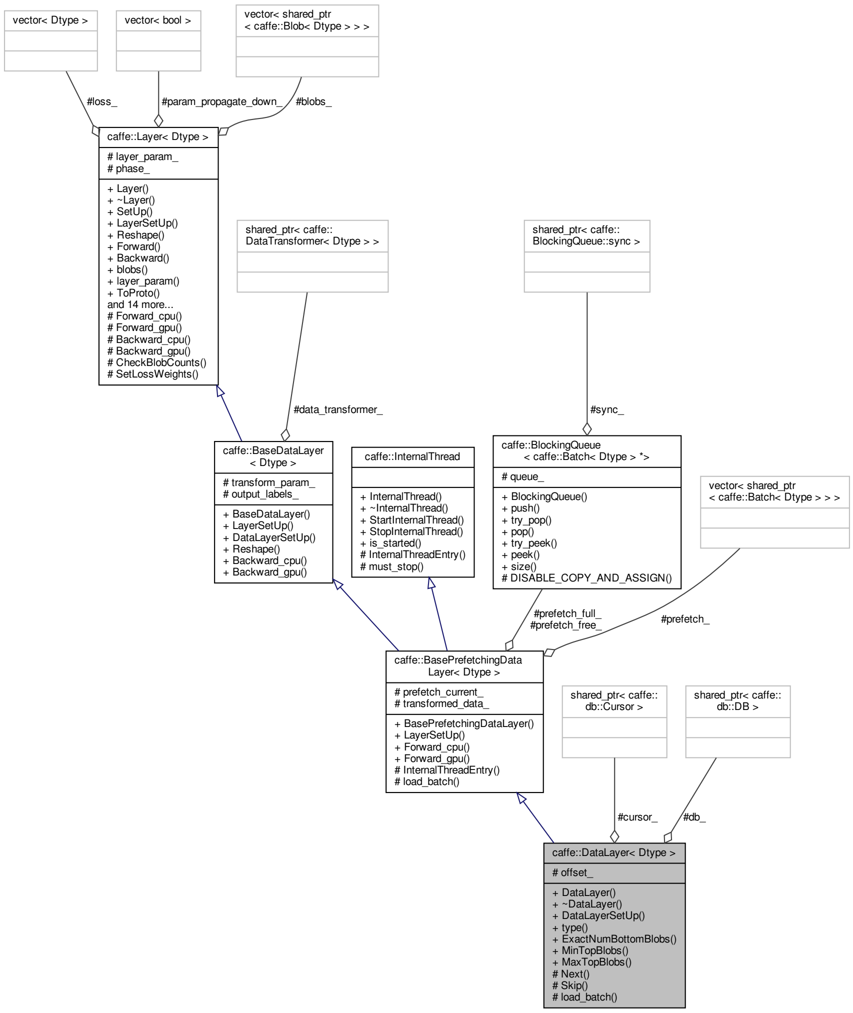
- include/caffe/util/db.hpp

5.29 caffe::DataLayer< Dtype > Class Template Reference

Inheritance diagram for caffe::DataLayer< Dtype >:



Collaboration diagram for caffe::DataLayer< Dtype >:



Public Member Functions

- **DataLayer** (const LayerParameter &param)
- virtual void **DataLayerSetUp** (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)
- virtual const char ∗ type () const

  Returns the layer type.
- virtual int ExactNumBottomBlobs () const

  Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.
- virtual int MinTopBlobs () const

  Returns the minimum number of top blobs required by the layer, or -1 if no minimum number is required.
- virtual int MaxTopBlobs () const

  Returns the maximum number of top blobs required by the layer, or -1 if no maximum number is required.

Protected Member Functions

- void **Next** ()
- bool **Skip** ()
- virtual void **load_batch** (Batch< Dtype > ∗batch)

Protected Attributes

- shared_ptr< db::DB > **db_**
- shared_ptr< db::Cursor > **cursor_**
- uint64_t **offset_**

5.29.1 Member Function Documentation

5.29.1.1 ExactNumBottomBlobs()

```
template<typename Dtype >
virtual int caffe::DataLayer< Dtype >::ExactNumBottomBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.29.1.2 MaxTopBlobs()

```
template<typename Dtype >
virtual int caffe::DataLayer< Dtype >::MaxTopBlobs ( ) const  [inline], [virtual]
```

Returns the maximum number of top blobs required by the layer, or -1 if no maximum number is required.

This method should be overridden to return a non-negative value if your layer expects some maximum number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.29.1.3 MinTopBlobs()

```
template<typename Dtype >
virtual int caffe::DataLayer< Dtype >::MinTopBlobs ( ) const  [inline], [virtual]
```

Returns the minimum number of top blobs required by the layer, or -1 if no minimum number is required.

This method should be overridden to return a non-negative value if your layer expects some minimum number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

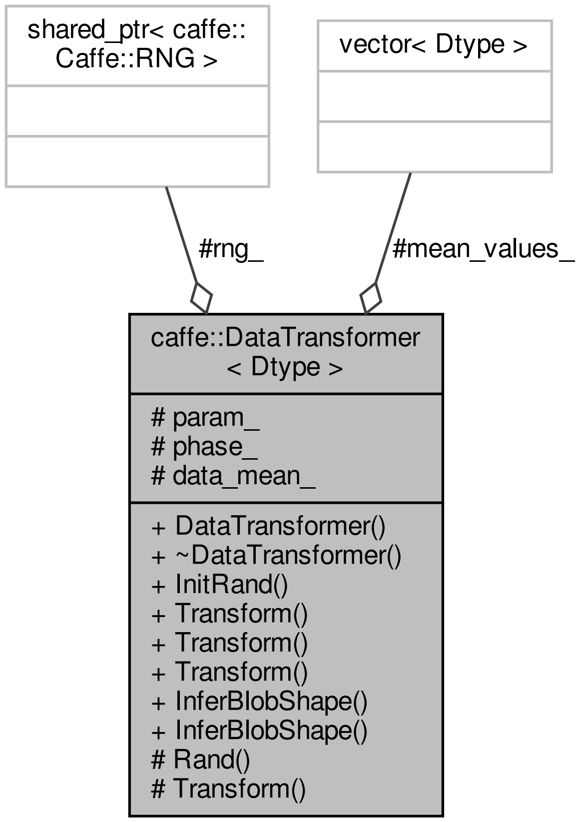
- include/caffe/layers/data_layer.hpp
- src/caffe/layers/data_layer.cpp

5.30 caffe::DataTransformer< Dtype > Class Template Reference

Applies common transformations to the input data, such as scaling, mirroring, substracting the image mean...

```
#include <data_transformer.hpp>
```

Collaboration diagram for caffe::DataTransformer< Dtype >:



Public Member Functions

- **DataTransformer** (const TransformationParameter &param, Phase phase)
- void InitRand ()

    Initialize the Random number generations if needed by the transformation.
- void Transform (const Datum &datum, Blob< Dtype > *transformed_blob)

    Applies the transformation defined in the data layer's transform_param block to the data.
- void Transform (const vector< Datum > &datum_vector, Blob< Dtype > *transformed_blob)

    Applies the transformation defined in the data layer's transform_param block to a vector of Datum.
- void Transform (Blob< Dtype > *input_blob, Blob< Dtype > *transformed_blob)

    Applies the same transformation defined in the data layer's transform_param block to all the num images in a input_blob.
- vector< int > InferBlobShape (const Datum &datum)

Infers the shape of transformed_blob will have when the transformation is applied to the data.

- vector< int > InferBlobShape (const vector< Datum > &datum_vector)

  Infers the shape of transformed_blob will have when the transformation is applied to the data. It uses the first element to infer the shape of the blob.

Protected Member Functions

- virtual int Rand (int n)

  Infers the shape of transformed_blob will have when the transformation is applied to the data. It uses the first element to infer the shape of the blob.
- void **Transform** (const Datum &datum, Dtype ∗transformed_data)

Protected Attributes

- TransformationParameter **param_**
- shared_ptr< Caffe::RNG > **rng_**
- Phase **phase_**
- Blob< Dtype > **data_mean_**
- vector< Dtype > **mean_values_**

5.30.1 Detailed Description

template<typename Dtype>
class caffe::DataTransformer< Dtype >

Applies common transformations to the input data, such as scaling, mirroring, substracting the image mean...

5.30.2 Member Function Documentation

5.30.2.1 InferBlobShape() [1/2]

```
template<typename Dtype >
vector< int > caffe::DataTransformer< Dtype >::InferBlobShape (
            const Datum & datum )
```

Infers the shape of transformed_blob will have when the transformation is applied to the data.

Parameters

datum	Datum containing the data to be transformed.

5.30.2.2 InferBlobShape() [2/2]

```
template<typename Dtype >
vector< int > caffe::DataTransformer< Dtype >::InferBlobShape (
            const vector< Datum > & datum_vector )
```

Infers the shape of transformed_blob will have when the transformation is applied to the data. It uses the first element to infer the shape of the blob.

Parameters

datum_vector	A vector of Datum containing the data to be transformed.

5.30.2.3 Rand()

```
template<typename Dtype >
int caffe::DataTransformer< Dtype >::Rand (
            int n ) [protected], [virtual]
```

Infers the shape of transformed_blob will have when the transformation is applied to the data. It uses the first element to infer the shape of the blob.

Parameters

mat_vector	A vector of Mat containing the data to be transformed. Generates a random integer from Uniform({0, 1, ..., n-1}).
n	The upperbound (exclusive) value of the random number.

Returns

A uniformly random integer value from ({0, 1, ..., n-1}).

5.30.2.4 Transform() [1/3]

```
template<typename Dtype >
void caffe::DataTransformer< Dtype >::Transform (
            const Datum & datum,
            Blob< Dtype > * transformed_blob )
```

Applies the transformation defined in the data layer's transform_param block to the data.

Parameters

datum	Datum containing the data to be transformed.
transformed_blob	This is destination blob. It can be part of top blob's data if set_cpu_data() is used. See data_layer.cpp for an example.

5.30.2.5 Transform() [2/3]

```
template<typename Dtype >
void caffe::DataTransformer< Dtype >::Transform (
            const vector< Datum > & datum_vector,
            Blob< Dtype > * transformed_blob )
```

Applies the transformation defined in the data layer's transform_param block to a vector of Datum.

Parameters

datum_vector	A vector of Datum containing the data to be transformed.
transformed_blob	This is destination blob. It can be part of top blob's data if set_cpu_data() is used. See memory_layer.cpp for an example.

5.30.2.6 Transform() [3/3]

```
template<typename Dtype >
void caffe::DataTransformer< Dtype >::Transform (
            Blob< Dtype > * input_blob,
            Blob< Dtype > * transformed_blob )
```

Applies the same transformation defined in the data layer's transform_param block to all the num images in a input_blob.

Parameters

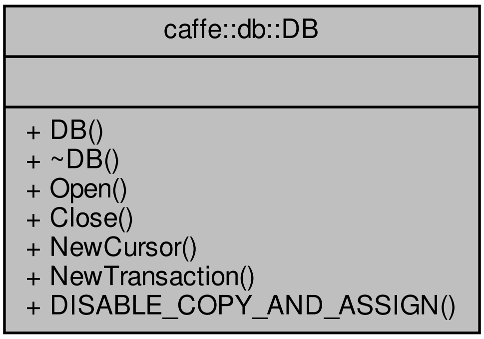
input_blob	A Blob containing the data to be transformed. It applies the same transformation to all the num images in the blob.
transformed_blob	This is destination blob, it will contain as many images as the input blob. It can be part of top blob's data.

The documentation for this class was generated from the following files:

- include/caffe/data_transformer.hpp
- src/caffe/data_transformer.cpp

5.31 caffe::db::DB Class Reference

Collaboration diagram for caffe::db::DB:



Public Member Functions

- virtual void **Open** (const string &source, Mode mode)=0
- virtual void **Close** ()=0
- virtual Cursor * **NewCursor** ()=0
- virtual Transaction * **NewTransaction** ()=0
- **DISABLE_COPY_AND_ASSIGN** (DB)

The documentation for this class was generated from the following file:

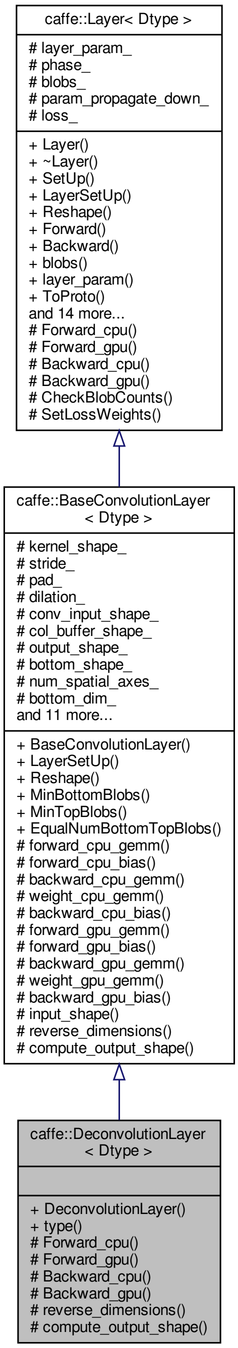
- include/caffe/util/db.hpp

5.32 caffe::DeconvolutionLayer< Dtype > Class Template Reference

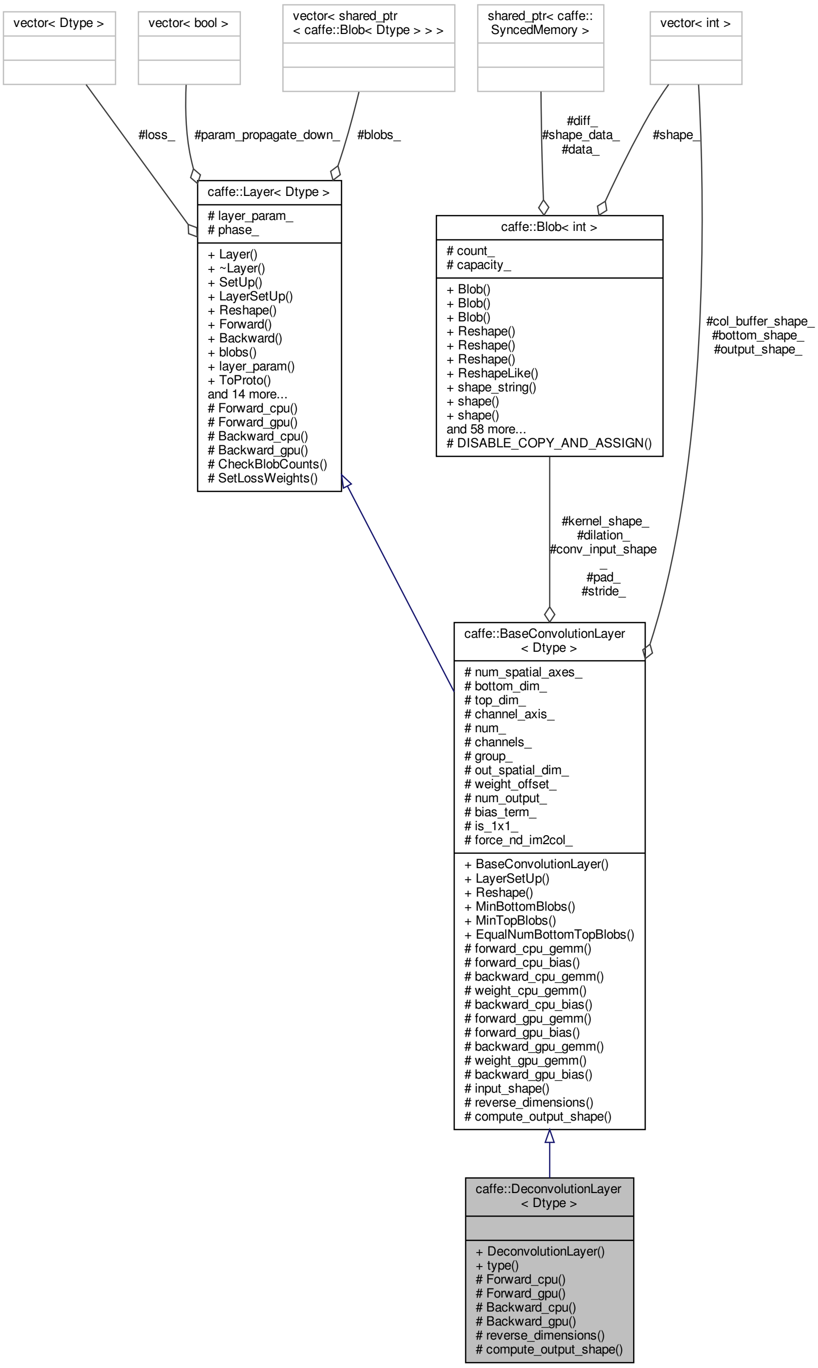
Convolve the input with a bank of learned filters, and (optionally) add biases, treating filters and convolution parameters in the opposite sense as ConvolutionLayer.

```
#include <deconv_layer.hpp>
```

Inheritance diagram for caffe::DeconvolutionLayer< Dtype >:



Collaboration diagram for caffe::DeconvolutionLayer< Dtype >:



Public Member Functions

- **DeconvolutionLayer** (const LayerParameter &param)
- virtual const char ∗ type () const

  Returns the layer type.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

    Using the CPU device, compute the layer output.
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

    Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

    Using the CPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

    Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.
- virtual bool **reverse_dimensions** ()
- virtual void **compute_output_shape** ()

Additional Inherited Members

5.32.1 Detailed Description

template< typename Dtype>
class caffe::DeconvolutionLayer< Dtype >

Convolve the input with a bank of learned filters, and (optionally) add biases, treating filters and convolution parameters in the opposite sense as ConvolutionLayer.

ConvolutionLayer computes each output value by dotting an input window with a filter; DeconvolutionLayer multiplies each input value by a filter elementwise, and sums over the resulting output windows. In other words, DeconvolutionLayer is ConvolutionLayer with the forward and backward passes reversed. DeconvolutionLayer reuses ConvolutionParameter for its parameters, but they take the opposite sense as in ConvolutionLayer (so padding is removed from the output rather than added to the input, and stride results in upsampling rather than downsampling).

The documentation for this class was generated from the following files:

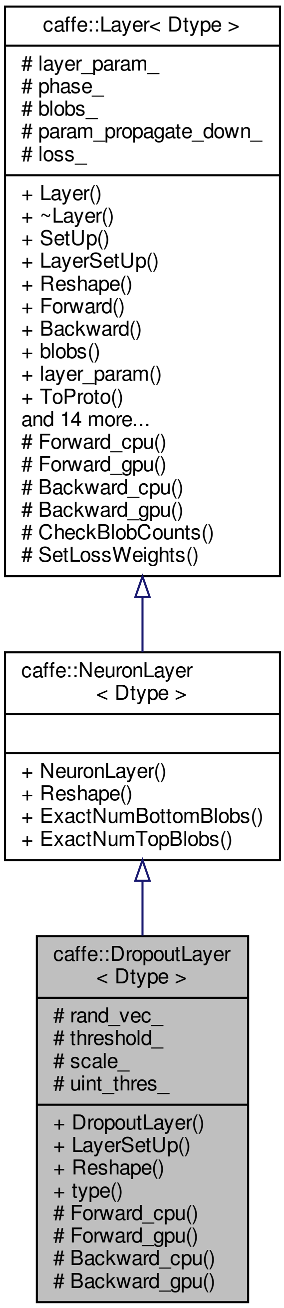
- include/caffe/layers/deconv_layer.hpp
- src/caffe/layers/deconv_layer.cpp

5.33 caffe::DropoutLayer< Dtype > Class Template Reference

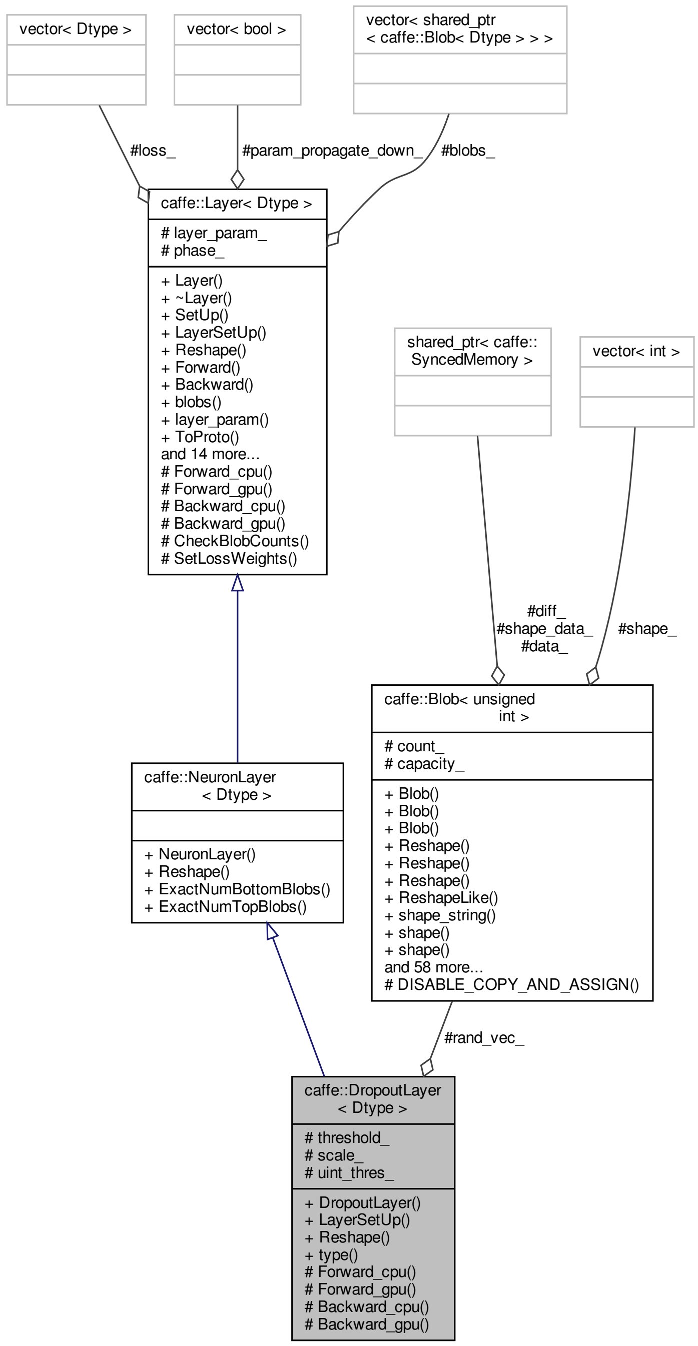
During training only, sets a random portion of x to 0, adjusting the rest of the vector magnitude accordingly.

```
#include <dropout_layer.hpp>
```

Inheritance diagram for caffe::DropoutLayer< Dtype >:



Collaboration diagram for caffe::DropoutLayer< Dtype >:



Public Member Functions

- DropoutLayer (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void Reshape (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

- virtual const char $*$ type () const

  Returns the layer type.

Protected Member Functions

- virtual void Forward_cpu (const vector$<$ Blob$<$ Dtype $>$ $*$$>$ &bottom, const vector$<$ Blob$<$ Dtype $>$ $*$$>$ &top)
- virtual void Forward_gpu (const vector$<$ Blob$<$ Dtype $>$ $*$$>$ &bottom, const vector$<$ Blob$<$ Dtype $>$ $*$$>$ &top)

  Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector$<$ Blob$<$ Dtype $>$ $*$$>$ &top, const vector$<$ bool $>$ &propagate_down, const vector$<$ Blob$<$ Dtype $>$ $*$$>$ &bottom)

  Using the CPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true.
- virtual void Backward_gpu (const vector$<$ Blob$<$ Dtype $>$ $*$$>$ &top, const vector$<$ bool $>$ &propagate_down, const vector$<$ Blob$<$ Dtype $>$ $*$$>$ &bottom)

  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Protected Attributes

- Blob$<$ unsigned int $>$ rand_vec_

  when divided by UINT_MAX, the randomly generated values $u \sim U(0, 1)$
- Dtype threshold_

  the probability p *of dropping any input*
- Dtype scale_

  the scale for undropped inputs at train time $1/(1-p)$
- unsigned int **uint_thres_**

5.33.1 Detailed Description

template$<$typename Dtype$>$
class caffe::DropoutLayer$<$ Dtype $>$

During training only, sets a random portion of x to 0, adjusting the rest of the vector magnitude accordingly.

Parameters

bottom	input Blob vector (length 1) 1. $(N \times C \times H \times W)$ the inputs x
top	output Blob vector (length 1) 1. $(N \times C \times H \times W)$ the computed outputs $y = \|x\|$

5.33.2 Constructor & Destructor Documentation

5.33.2.1 DropoutLayer()

```
template<typename Dtype >
caffe::DropoutLayer< Dtype >::DropoutLayer (
            const LayerParameter & param )  [inline], [explicit]
```

Parameters

param	provides DropoutParameter dropout_param, with DropoutLayer options: • dropout_ratio (**optional**, default 0.5). Sets the probability p that any given unit is dropped.

5.33.3 Member Function Documentation

5.33.3.1 Forward_cpu()

```
template<typename Dtype >
void caffe::DropoutLayer< Dtype >::Forward_cpu (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [protected], [virtual]
```

Parameters

bottom	input Blob vector (length 1) 1. $(N \times C \times H \times W)$ the inputs x
top	output Blob vector (length 1) 1. $(N \times C \times H \times W)$ the computed outputs. At training time, we have $y_{\mbox{train}} = \left\{ \begin{array}{ll} \frac{x}{1 - p} & \mbox{if } u > p \\ 0 & \mbox{otherwise} \end{array} \right.$, where $u \sim U(0, 1)$ is generated independently for each input at each iteration. At test time, we simply have $y_{\mbox{test}} = \mathbb{E}[y_{\mbox{train}}] = x$.

Implements caffe::Layer< Dtype >.

5.33.3.2 LayerSetUp()

```
template<typename Dtype >
void caffe::DropoutLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

5.33.3.3 Reshape()

```
template<typename Dtype >
void caffe::DropoutLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Reimplemented from caffe::NeuronLayer< Dtype >.

The documentation for this class was generated from the following files:

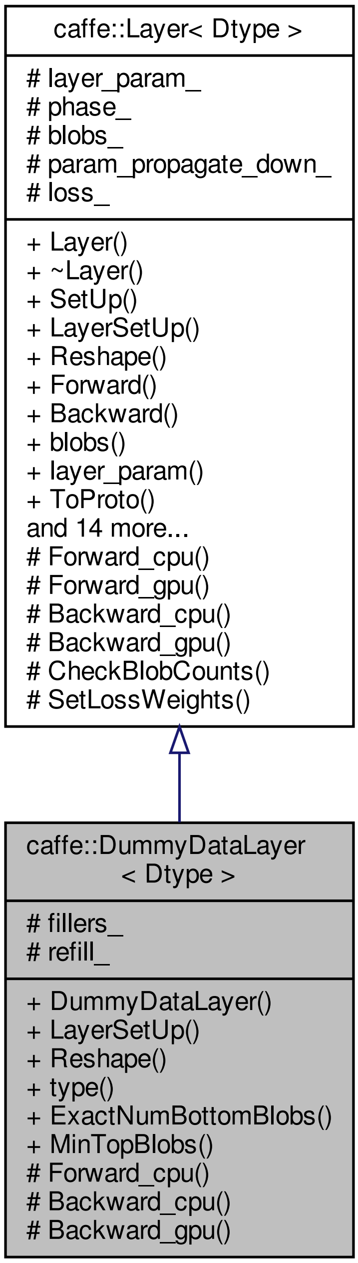
- include/caffe/layers/dropout_layer.hpp
- src/caffe/layers/dropout_layer.cpp

5.34 caffe::DummyDataLayer< Dtype > Class Template Reference

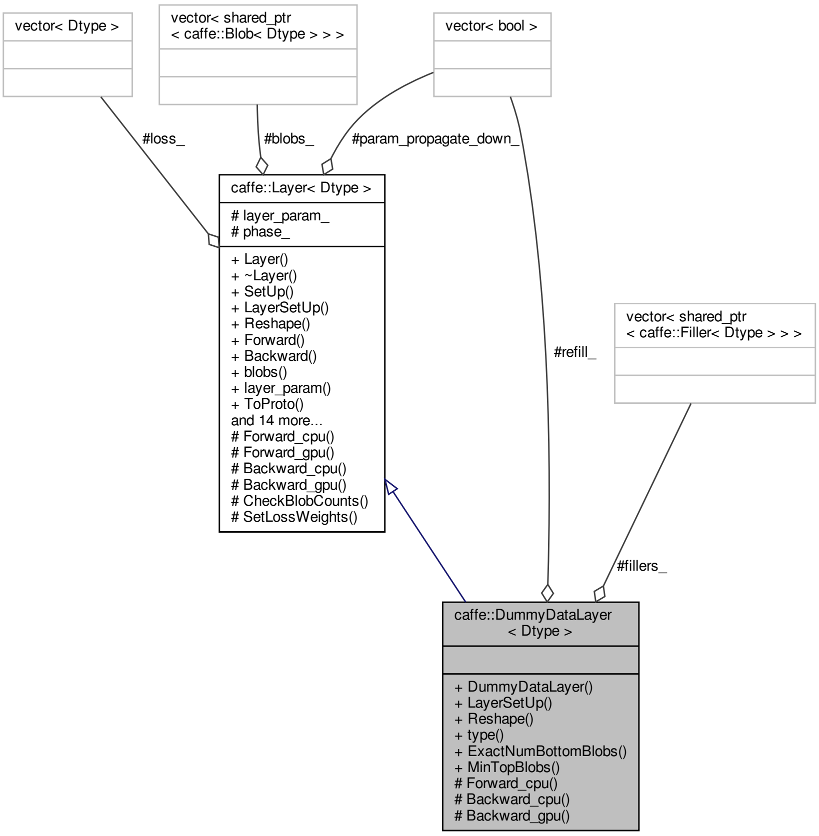
Provides data to the Net generated by a Filler.

`#include <dummy_data_layer.hpp>`

Inheritance diagram for caffe::DummyDataLayer< Dtype >:



Collaboration diagram for caffe::DummyDataLayer< Dtype >:



Public Member Functions

- **DummyDataLayer** (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

    Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void Reshape (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

    Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.
- virtual const char ∗ type () const

    Returns the layer type.
- virtual int ExactNumBottomBlobs () const

    Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.
- virtual int MinTopBlobs () const

    Returns the minimum number of top blobs required by the layer, or -1 if no minimum number is required.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

    Using the CPU device, compute the layer output.

- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the CPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Protected Attributes

- vector< shared_ptr< Filler< Dtype > > > **fillers_**
- vector< bool > **refill_**

5.34.1 Detailed Description

template<typename Dtype>
class caffe::DummyDataLayer< Dtype >

Provides data to the Net generated by a Filler.

TODO(dox): thorough documentation for Forward and proto params.

5.34.2 Member Function Documentation

5.34.2.1 ExactNumBottomBlobs()

```
template<typename Dtype >
virtual int caffe::DummyDataLayer< Dtype >::ExactNumBottomBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.34.2.2 LayerSetUp()

```
template<typename Dtype >
void caffe::DummyDataLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

5.34.2.3 MinTopBlobs()

```
template<typename Dtype >
virtual int caffe::DummyDataLayer< Dtype >::MinTopBlobs ( ) const  [inline], [virtual]
```

Returns the minimum number of top blobs required by the layer, or -1 if no minimum number is required.

This method should be overridden to return a non-negative value if your layer expects some minimum number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.34.2.4 Reshape()

```
template<typename Dtype >
virtual void caffe::DummyDataLayer< Dtype >::Reshape (
             const vector< Blob< Dtype > *> & bottom,
             const vector< Blob< Dtype > *> & top )  [inline], [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

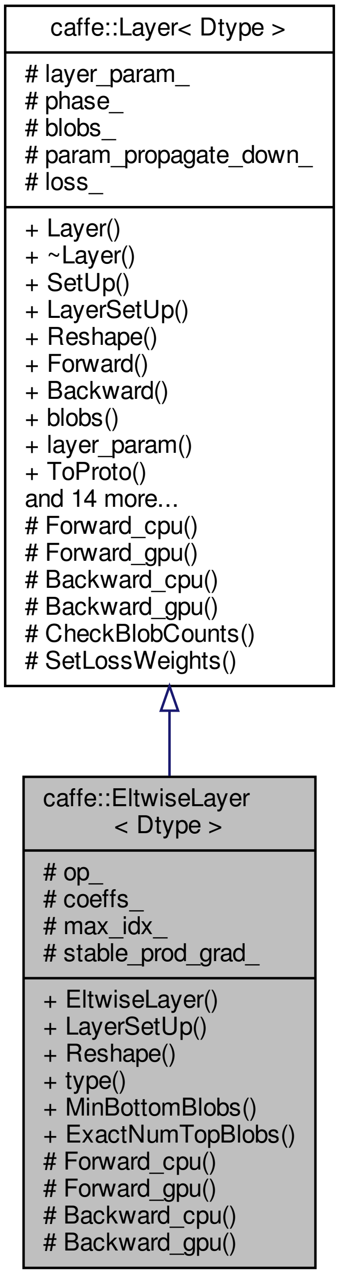
- include/caffe/layers/dummy_data_layer.hpp
- src/caffe/layers/dummy_data_layer.cpp

5.35 caffe::EltwiseLayer< Dtype > Class Template Reference

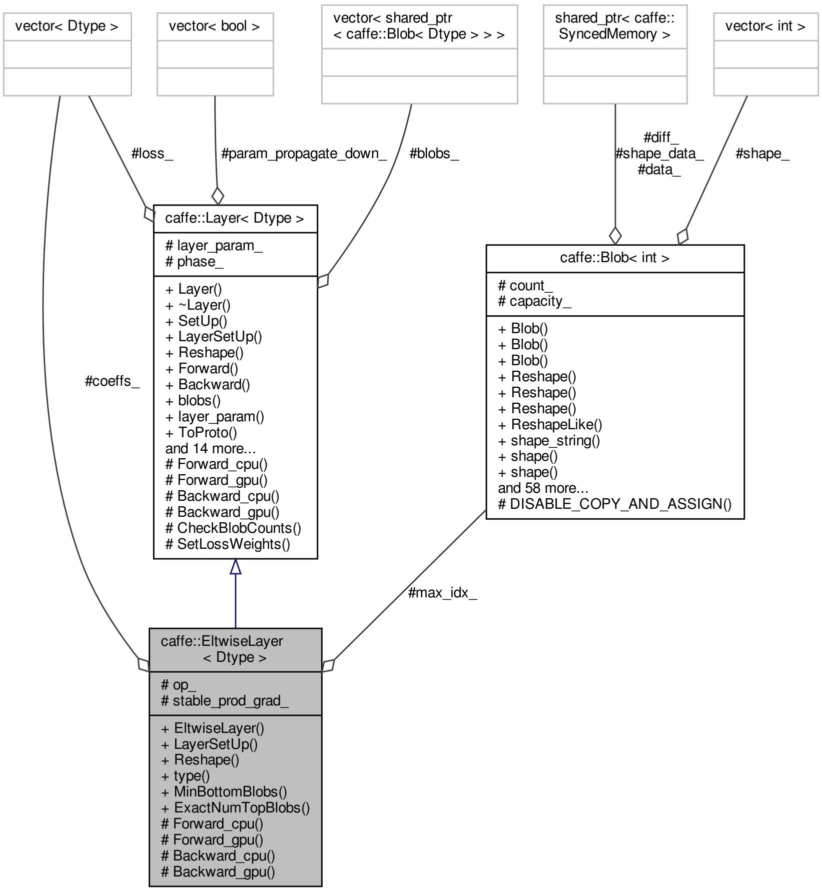
Compute elementwise operations, such as product and sum, along multiple input Blobs.

```
#include <eltwise_layer.hpp>
```

Inheritance diagram for caffe::EltwiseLayer< Dtype >:



Collaboration diagram for caffe::EltwiseLayer< Dtype >:



Public Member Functions

- **EltwiseLayer** (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void Reshape (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.
- virtual const char ∗ type () const

  Returns the layer type.
- virtual int MinBottomBlobs () const

  Returns the minimum number of bottom blobs required by the layer, or -1 if no minimum number is required.
- virtual int ExactNumTopBlobs () const

  Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

Using the CPU device, compute the layer output.
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the CPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Protected Attributes

- EltwiseParameter_EltwiseOp **op_**
- vector< Dtype > **coeffs_**
- Blob< int > **max_idx_**
- bool **stable_prod_grad_**

5.35.1 Detailed Description

template<typename Dtype>
class caffe::EltwiseLayer< Dtype >

Compute elementwise operations, such as product and sum, along multiple input Blobs.

TODO(dox): thorough documentation for Forward, Backward, and proto params.

5.35.2 Member Function Documentation

5.35.2.1 ExactNumTopBlobs()

```
template<typename Dtype >
virtual int caffe::EltwiseLayer< Dtype >::ExactNumTopBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.35.2.2 LayerSetUp()

```
template<typename Dtype >
void caffe::EltwiseLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

5.35.2.3 MinBottomBlobs()

```
template<typename Dtype >
virtual int caffe::EltwiseLayer< Dtype >::MinBottomBlobs ( ) const  [inline], [virtual]
```

Returns the minimum number of bottom blobs required by the layer, or -1 if no minimum number is required.

This method should be overridden to return a non-negative value if your layer expects some minimum number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.35.2.4 Reshape()

```
template<typename Dtype >
void caffe::EltwiseLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

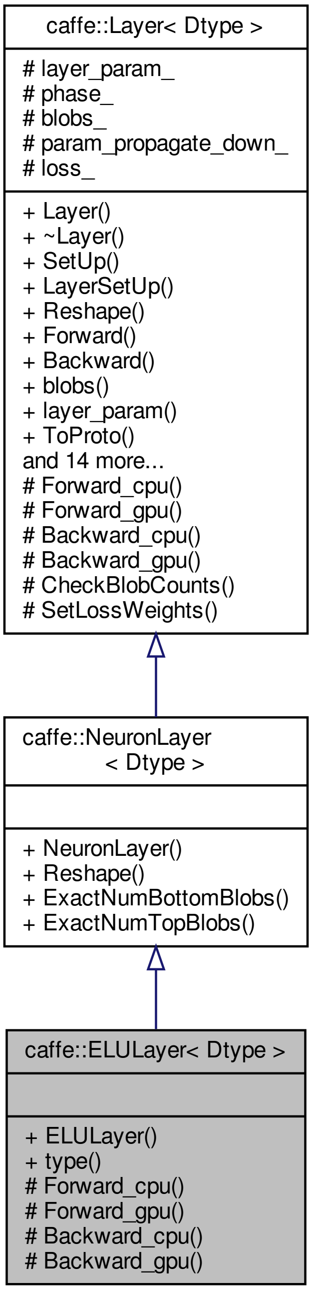
- include/caffe/layers/eltwise_layer.hpp
- src/caffe/layers/eltwise_layer.cpp

5.36 caffe::ELULayer< Dtype > Class Template Reference

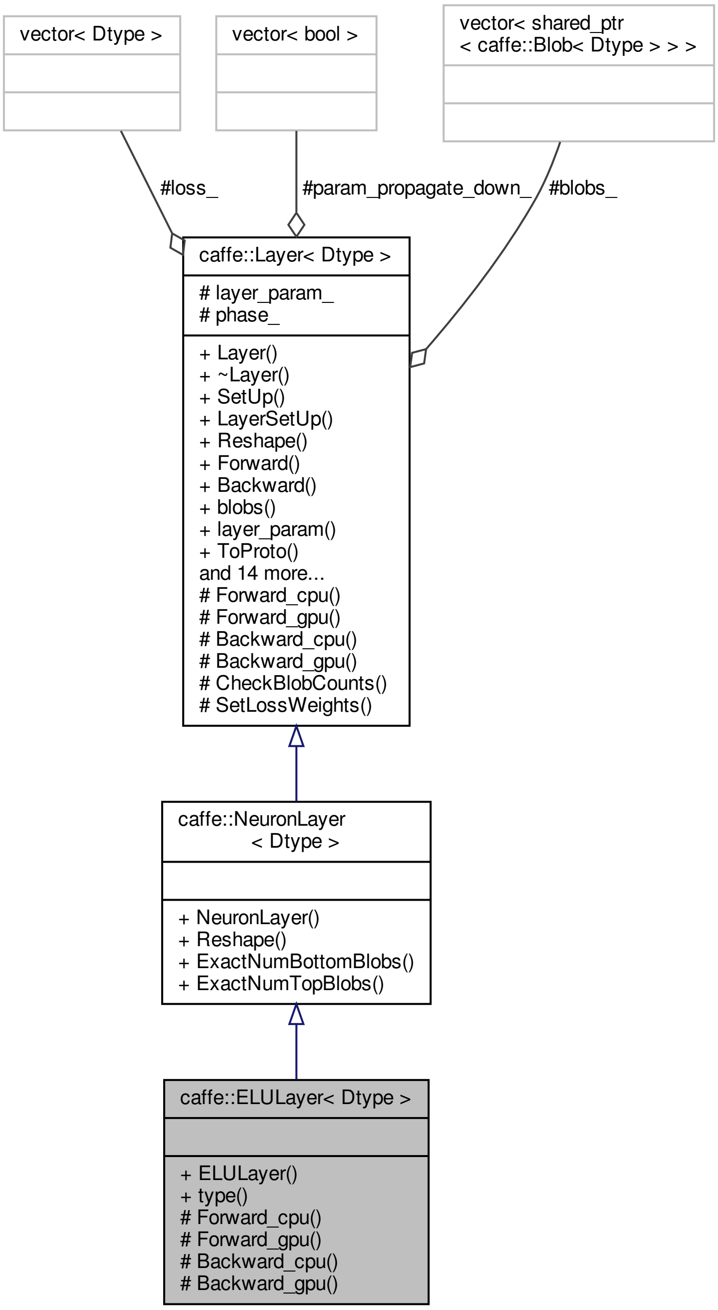
Exponential Linear Unit non-linearity $y = \left\{ \begin{array}{lr} x & \mathrm{if} \; x > 0 \\ \alpha (\exp(x)-1) & \mathrm{if} \; x \le 0 \end{array} \right.$.

```
#include <elu_layer.hpp>
```

Inheritance diagram for caffe::ELULayer< Dtype >:



Collaboration diagram for caffe::ELULayer< Dtype >:



Public Member Functions

- ELULayer (const LayerParameter &param)
- virtual const char ∗ type () const

  Returns the layer type.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Computes the error gradient w.r.t. the ELU inputs.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Additional Inherited Members

5.36.1 Detailed Description

template<typename Dtype>
class caffe::ELULayer< Dtype >

Exponential Linear Unit non-linearity $y = \left\{ \begin{array}{lr} x & \mathrm{if} \; x > 0 \\ \alpha (\exp(x)-1) & \mathrm{if} \; x \le 0 \end{array} \right.$.

5.36.2 Constructor & Destructor Documentation

5.36.2.1 ELULayer()

```
template<typename Dtype >
caffe::ELULayer< Dtype >::ELULayer (
            const LayerParameter & param )  [inline], [explicit]
```

Parameters

param	provides ELUParameter elu_param, with ELULayer options: • alpha (**optional**, default 1). the value α by which controls saturation for negative inputs.

5.36.3 Member Function Documentation

5.36.3.1 Backward_cpu()

```
template<typename Dtype >
void caffe::ELULayer< Dtype >::Backward_cpu (
            const vector< Blob< Dtype > *> & top,
            const vector< bool > & propagate_down,
            const vector< Blob< Dtype > *> & bottom )  [protected], [virtual]
```

Computes the error gradient w.r.t. the ELU inputs.

Parameters

top	output Blob vector (length 1), providing the error gradient with respect to the outputs 1. $(N \times C \times H \times W)$ containing error gradients $\frac{\partial E}{\partial y}$ with respect to computed outputs y
propagate_down	see Layer::Backward.
bottom	input Blob vector (length 1) 1. $(N \times C \times H \times W)$ the inputs x; Backward fills their diff with gradients $\frac{\partial E}{\partial x} = \left\{ \begin{array}{lr} 1 & \mathrm{if} \; x > 0 \\ y + \alpha & \mathrm{if} \; x \le 0 \end{array} \right.$ if propagate_down[0].

Implements caffe::Layer< Dtype >.

5.36.3.2 Forward_cpu()

```
template<typename Dtype >
void caffe::ELULayer< Dtype >::Forward_cpu (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [protected], [virtual]
```

Parameters

bottom	input Blob vector (length 1) 1. $(N \times C \times H \times W)$ the inputs x
top	output Blob vector (length 1) 1. $(N \times C \times H \times W)$ the computed outputs $y = \left\{ \begin{array}{lr} x & \mathrm{if} \; x > 0 \\ \alpha (\exp(x)-1) & \mathrm{if} \; x \le 0 \end{array} \right.$.

Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

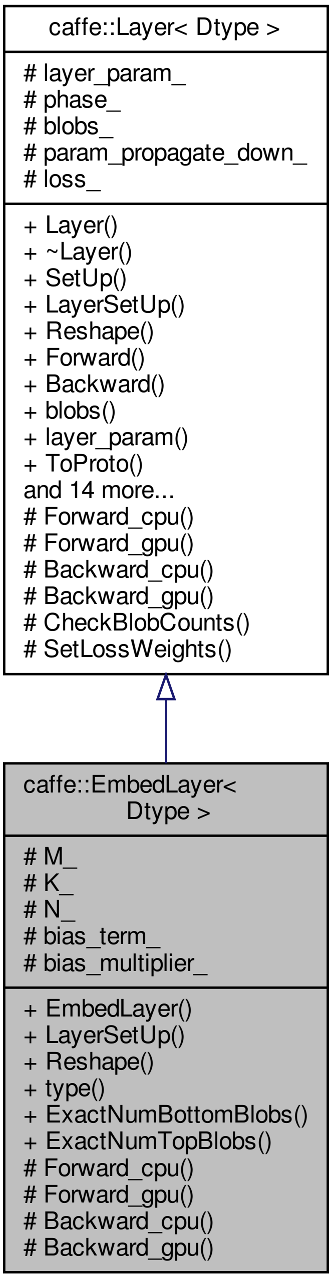
- include/caffe/layers/elu_layer.hpp
- src/caffe/layers/elu_layer.cpp

5.37 caffe::EmbedLayer< Dtype > Class Template Reference

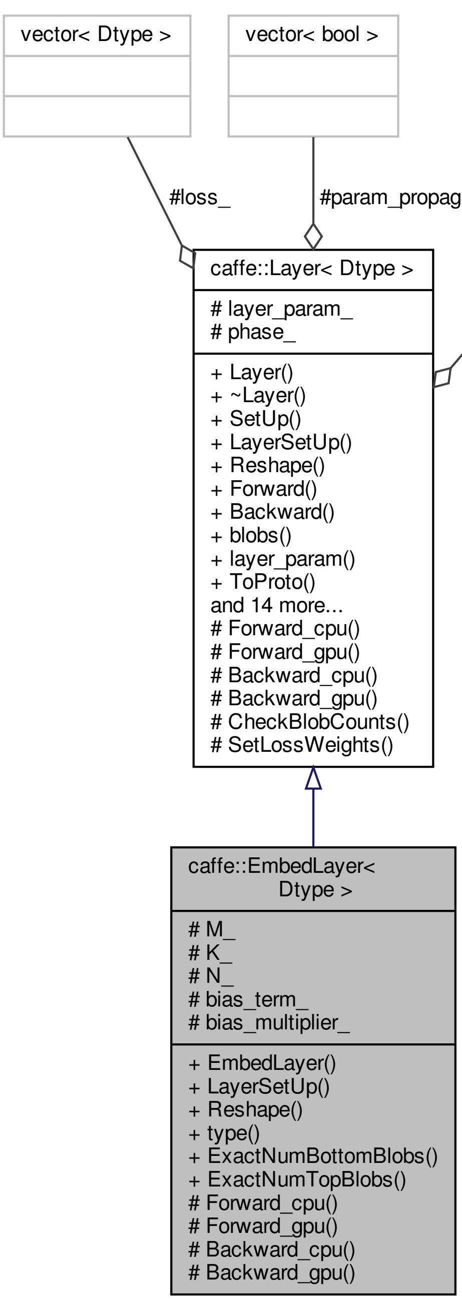
A layer for learning "embeddings" of one-hot vector input. Equivalent to an InnerProductLayer with one-hot vectors as input, but for efficiency the input is the "hot" index of each column itself.

```
#include <embed_layer.hpp>
```

Inheritance diagram for caffe::EmbedLayer< Dtype >:



Collaboration diagram for caffe::EmbedLayer< Dtype >:



Public Member Functions

- **EmbedLayer** (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

    Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void Reshape (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

    Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

- virtual const char $*$ type () const

  Returns the layer type.
- virtual int ExactNumBottomBlobs () const

  Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.
- virtual int ExactNumTopBlobs () const

  Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > $*$> &bottom, const vector< Blob< Dtype > $*$> &top)

  Using the CPU device, compute the layer output.
- virtual void Forward_gpu (const vector< Blob< Dtype > $*$> &bottom, const vector< Blob< Dtype > $*$> &top)

  Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > $*$> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > $*$> &bottom)

  Using the CPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true.
- virtual void Backward_gpu (const vector< Blob< Dtype > $*$> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > $*$> &bottom)

  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Protected Attributes

- int **M_**
- int **K_**
- int **N_**
- bool **bias_term_**
- Blob< Dtype > **bias_multiplier_**

5.37.1 Detailed Description

template<typename Dtype>
class caffe::EmbedLayer< Dtype >

A layer for learning "embeddings" of one-hot vector input. Equivalent to an InnerProductLayer with one-hot vectors as input, but for efficiency the input is the "hot" index of each column itself.

TODO(dox): thorough documentation for Forward, Backward, and proto params.

5.37.2 Member Function Documentation

5.37.2.1 ExactNumBottomBlobs()

```
template<typename Dtype >
virtual int caffe::EmbedLayer< Dtype >::ExactNumBottomBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.37.2.2 ExactNumTopBlobs()

```
template<typename Dtype >
virtual int caffe::EmbedLayer< Dtype >::ExactNumTopBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.37.2.3 LayerSetUp()

```
template<typename Dtype >
void caffe::EmbedLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

5.37.2.4 Reshape()

```
template<typename Dtype >
void caffe::EmbedLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

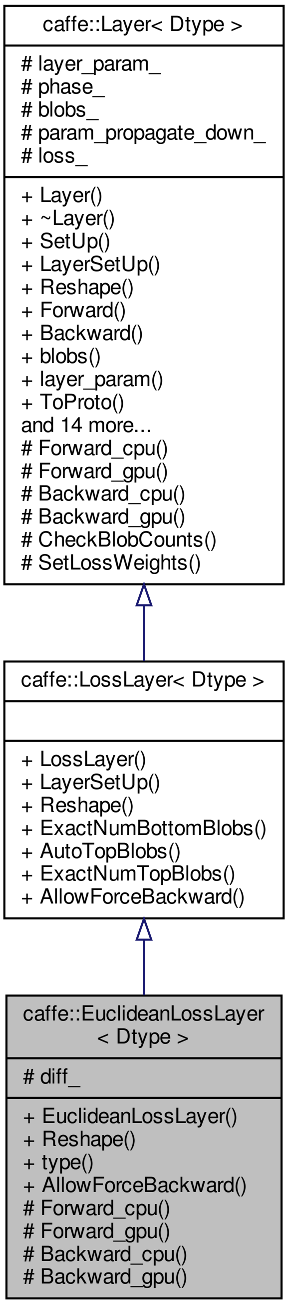
- include/caffe/layers/embed_layer.hpp
- src/caffe/layers/embed_layer.cpp

5.38 caffe::EuclideanLossLayer< Dtype > Class Template Reference

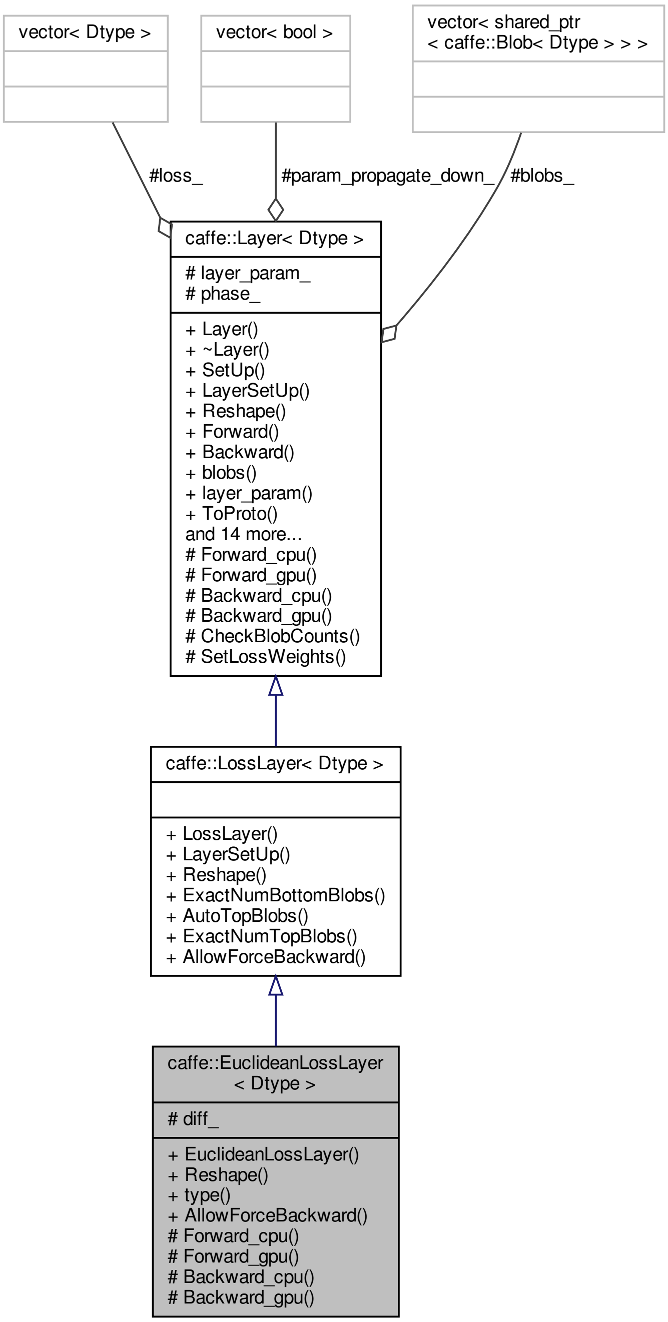
Computes the Euclidean (L2) loss $E = \frac{1}{2N} \sum\limits_{n=1}^N \left| \left| \hat{y}_n - y_n \right| \right|_2^2$ for real-valued regression tasks.

```
#include <euclidean_loss_layer.hpp>
```

Inheritance diagram for caffe::EuclideanLossLayer< Dtype >:



Collaboration diagram for caffe::EuclideanLossLayer< Dtype >:



Public Member Functions

- **EuclideanLossLayer** (const LayerParameter &param)
- virtual void Reshape (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.
- virtual const char ∗ type () const

  Returns the layer type.
- virtual bool AllowForceBackward (const int bottom_index) const

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

    Computes the Euclidean (L2) loss $E = \frac{1}{2N} \sum\limits_{n=1}^N \left| \left| \hat{y}_n - y_n \right| \right|_2^2$ for real-valued regression tasks.
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

    Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

    Computes the Euclidean error gradient w.r.t. the inputs.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

    Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Protected Attributes

- Blob< Dtype > **diff_**

5.38.1 Detailed Description

template<typename Dtype>
class caffe::EuclideanLossLayer< Dtype >

Computes the Euclidean (L2) loss $E = \frac{1}{2N} \sum\limits_{n=1}^N \left| \left| \hat{y}_n - y_n \right| \right|_2^2$ for real-valued regression tasks.

Parameters

bottom	input Blob vector (length 2) 1. $(N \times C \times H \times W)$ the predictions $\hat{y} \in [-\infty, +\infty]$ 2. $(N \times C \times H \times W)$ the targets $y \in [-\infty, +\infty]$
top	output Blob vector (length 1) 1. $(1 \times 1 \times 1 \times 1)$ the computed Euclidean loss: $E = \frac{1}{2n} \sum\limits_{n=1}^N \left| \left| \hat{y}_n - y_n \right| \right|_2^2$

This can be used for least-squares regression tasks. An InnerProductLayer input to a EuclideanLossLayer exactly formulates a linear least squares regression problem. With non-zero weight decay the problem becomes one of ridge regression – see src/caffe/test/test_gradient_based_solver.cpp for a concrete example wherein we check that the gradients computed for a Net with exactly this structure match hand-computed gradient formulas for ridge regression.

(Note: Caffe, and SGD in general, is certainly **not** the best way to solve linear least squares problems! We use it only as an instructive example.)

5.38.2 Member Function Documentation

5.38.2.1 AllowForceBackward()

```
template<typename Dtype >
virtual bool caffe::EuclideanLossLayer< Dtype >::AllowForceBackward (
              const int bottom_index ) const  [inline], [virtual]
```

Unlike most loss layers, in the EuclideanLossLayer we can backpropagate to both inputs – override to return true and always allow force_backward.

Reimplemented from caffe::LossLayer< Dtype >.

5.38.2.2 Backward_cpu()

```
template<typename Dtype >
void caffe::EuclideanLossLayer< Dtype >::Backward_cpu (
              const vector< Blob< Dtype > *> & top,
              const vector< bool > & propagate_down,
              const vector< Blob< Dtype > *> & bottom )  [protected], [virtual]
```

Computes the Euclidean error gradient w.r.t. the inputs.

Unlike other children of LossLayer, EuclideanLossLayer **can** compute gradients with respect to the label inputs bottom[1] (but still only will if propagate_down[1] is set, due to being produced by learnable parameters or if force↵_backward is set). In fact, this layer is "commutative" – the result is the same regardless of the order of the two bottoms.

Parameters

top	output Blob vector (length 1), providing the error gradient with respect to the outputs 1. $(1 \times 1 \times 1 \times 1)$ This Blob's diff will simply contain the loss_weight$*$ λ, as λ is the coefficient of this layer's output ℓ_i in the overall Net loss $E = \lambda_i \ell_i +$ other loss terms; hence $\frac{\partial E}{\partial \ell_i} = \lambda_i$. ($*$Assuming that this top Blob is not used as a bottom (input) by any other layer of the Net.)
propagate_down	see Layer::Backward.
bottom	input Blob vector (length 2) 1. $(N \times C \times H \times W)$ the predictions $\hat{y}$; Backward fills their diff with gradients $\frac{\partial E}{\partial \hat{y}} = \frac{1}{n} \sum\limits_{n=1}^{N} (\hat{y}_n - y_n)$ if propagate_down[0] 2. $(N \times C \times H \times W)$ the targets y; Backward fills their diff with gradients $\frac{\partial E}{\partial y} = \frac{1}{n} \sum\limits_{n=1}^{N} (y_n - \hat{y}_n)$ if propagate_down[1]

Implements caffe::Layer< Dtype >.

5.38.2.3 Forward_cpu()

```
template<typename Dtype >
void caffe::EuclideanLossLayer< Dtype >::Forward_cpu (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [protected], [virtual]
```

Computes the Euclidean (L2) loss $E = \frac{1}{2N} \sum\limits_{n=1}^N \left| \left| \hat{y}_n - y_n \right| \right|_2^2$ for real-valued regression tasks.

Parameters

bottom	input Blob vector (length 2) 1. $(N \times C \times H \times W)$ the predictions $\hat{y} \in [-\infty, +\infty]$ 2. $(N \times C \times H \times W)$ the targets $y \in [-\infty, +\infty]$
top	output Blob vector (length 1) 1. $(1 \times 1 \times 1 \times 1)$ the computed Euclidean loss: $E = \frac{1}{2n} \sum\limits_{n=1}^N \left| \left| \hat{y}_n - y_n \right| \right|_2^2$

This can be used for least-squares regression tasks. An InnerProductLayer input to a EuclideanLossLayer exactly formulates a linear least squares regression problem. With non-zero weight decay the problem becomes one of ridge regression – see src/caffe/test/test_gradient_based_solver.cpp for a concrete example wherein we check that the gradients computed for a Net with exactly this structure match hand-computed gradient formulas for ridge regression.

(Note: Caffe, and SGD in general, is certainly **not** the best way to solve linear least squares problems! We use it only as an instructive example.)

Implements caffe::Layer< Dtype >.

5.38.2.4 Reshape()

```
template<typename Dtype >
void caffe::EuclideanLossLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as

reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Reimplemented from caffe::LossLayer< Dtype >.

The documentation for this class was generated from the following files:

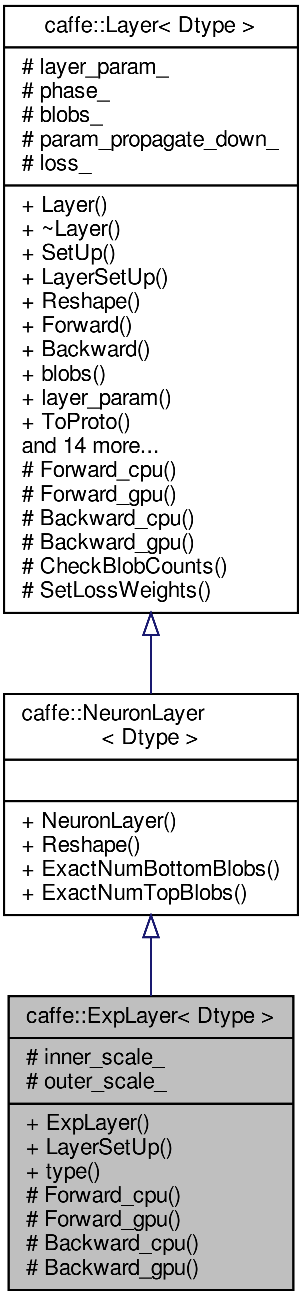
- include/caffe/layers/euclidean_loss_layer.hpp
- src/caffe/layers/euclidean_loss_layer.cpp

5.39 caffe::ExpLayer< Dtype > Class Template Reference

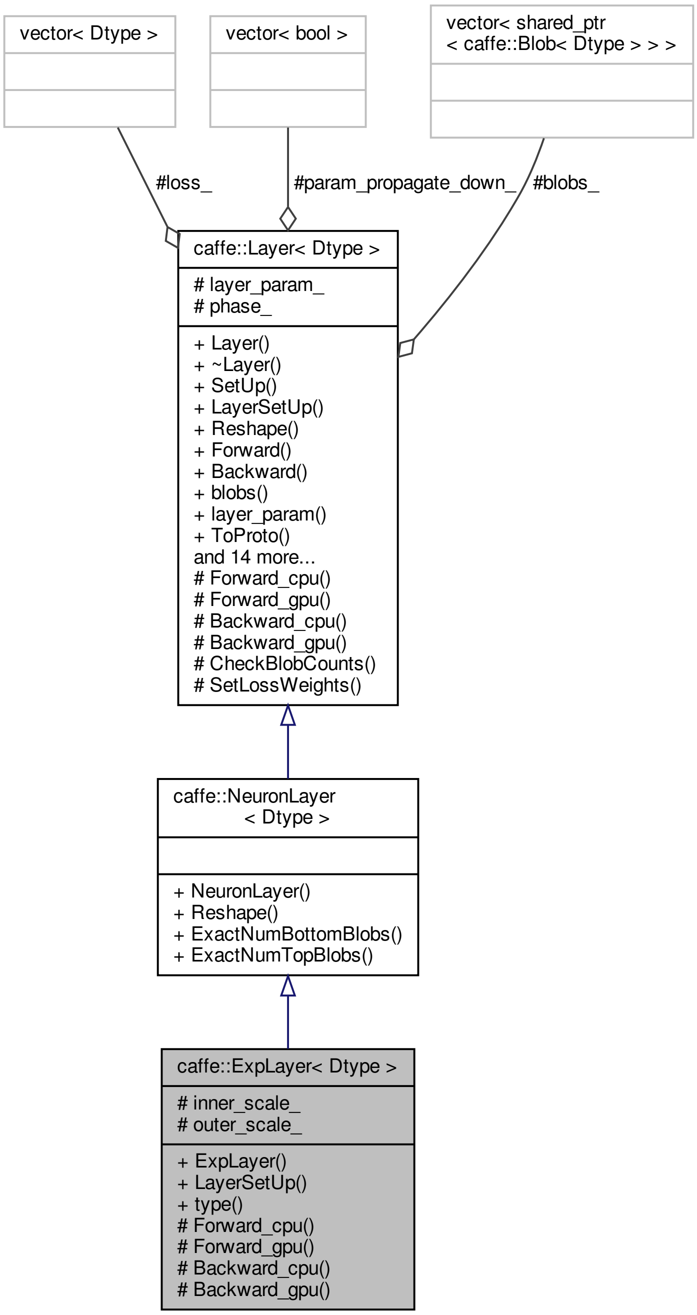
Computes $y = \gamma^{\alpha x + \beta}$, as specified by the scale α, shift β, and base γ.

`#include <exp_layer.hpp>`

Inheritance diagram for caffe::ExpLayer< Dtype >:



Collaboration diagram for caffe::ExpLayer< Dtype >:



Public Member Functions

- ExpLayer (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual const char ∗ type () const

  Returns the layer type.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

    Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

    Computes the error gradient w.r.t. the exp inputs.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

    Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Protected Attributes

- Dtype **inner_scale_**
- Dtype **outer_scale_**

5.39.1 Detailed Description

template<typename Dtype>
class caffe::ExpLayer< Dtype >

Computes $y = \gamma^{\alpha x + \beta}$, as specified by the scale α, shift β, and base γ.

5.39.2 Constructor & Destructor Documentation

5.39.2.1 ExpLayer()

```
template<typename Dtype >
caffe::ExpLayer< Dtype >::ExpLayer (
            const LayerParameter & param )  [inline], [explicit]
```

Parameters

param	provides ExpParameter exp_param, with ExpLayer options: • scale (**optional**, default 1) the scale α • shift (**optional**, default 0) the shift β • base (**optional**, default -1 for a value of $e \approx 2.718$) the base γ

5.39.3 Member Function Documentation

5.39.3.1 Backward_cpu()

```
template<typename Dtype >
void caffe::ExpLayer< Dtype >::Backward_cpu (
            const vector< Blob< Dtype > *> & top,
            const vector< bool > & propagate_down,
            const vector< Blob< Dtype > *> & bottom )  [protected], [virtual]
```

Computes the error gradient w.r.t. the exp inputs.

Parameters

top	output Blob vector (length 1), providing the error gradient with respect to the outputs 1. $(N \times C \times H \times W)$ containing error gradients $\frac{\partial E}{\partial y}$ with respect to computed outputs y
propagate_down	see Layer::Backward.
bottom	input Blob vector (length 1) 1. $(N \times C \times H \times W)$ the inputs x; Backward fills their diff with gradients $\frac{\partial E}{\partial x} = \frac{\partial E}{\partial y} y \alpha \log_e(gamma)$ if propagate_down[0]

Implements caffe::Layer< Dtype >.

5.39.3.2 Forward_cpu()

```
template<typename Dtype >
void caffe::ExpLayer< Dtype >::Forward_cpu (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [protected], [virtual]
```

Parameters

bottom	input Blob vector (length 1) 1. $(N \times C \times H \times W)$ the inputs x
top	output Blob vector (length 1) 1. $(N \times C \times H \times W)$ the computed outputs $y = \gamma^{\alpha x + \beta}$

Implements caffe::Layer< Dtype >.

5.39.3.3 LayerSetUp()

```
template<typename Dtype >
void caffe::ExpLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

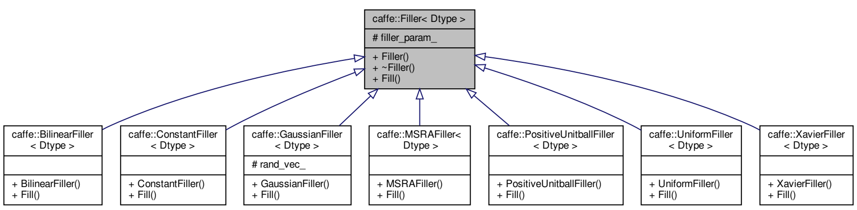
- include/caffe/layers/exp_layer.hpp
- src/caffe/layers/exp_layer.cpp

5.40 caffe::Filler< Dtype > Class Template Reference

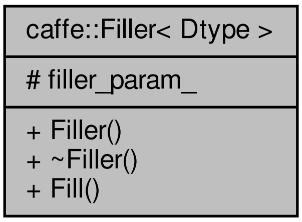
Fills a Blob with constant or randomly-generated data.

`#include <filler.hpp>`

Inheritance diagram for caffe::Filler< Dtype >:



Collaboration diagram for caffe::Filler< Dtype >:



Public Member Functions

- **Filler** (const FillerParameter &param)
- virtual void **Fill** (Blob< Dtype > ∗blob)=0

Protected Attributes

- FillerParameter **filler_param_**

5.40.1 Detailed Description

template<typename Dtype>
class caffe::Filler< Dtype >

Fills a Blob with constant or randomly-generated data.

The documentation for this class was generated from the following file:

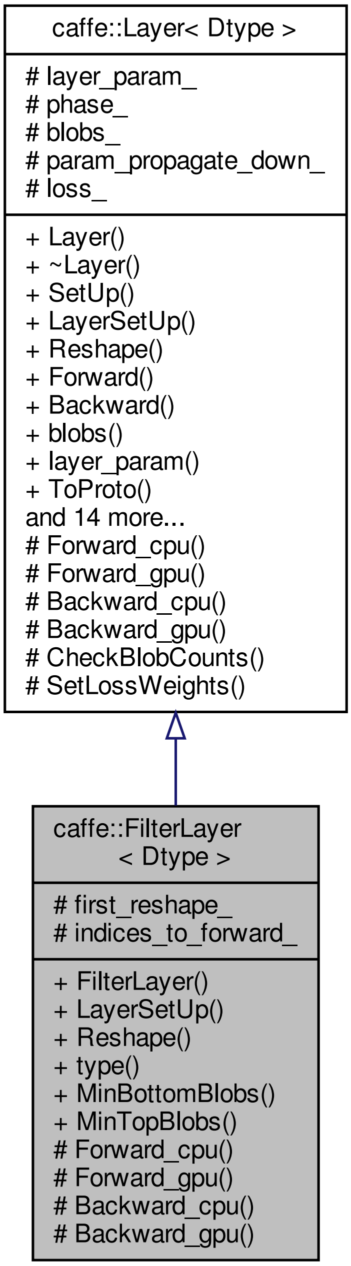
- include/caffe/filler.hpp

5.41 caffe::FilterLayer< Dtype > Class Template Reference

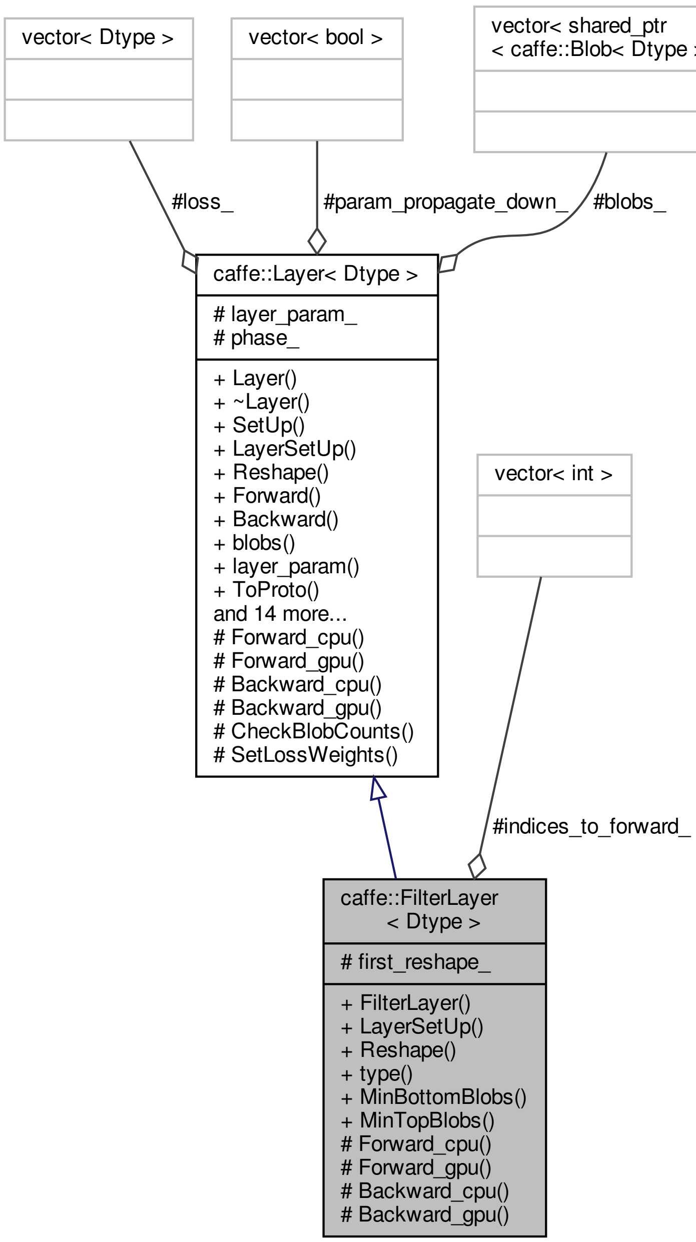
Takes two+ Blobs, interprets last Blob as a selector and filter remaining Blobs accordingly with selector data (0 means that the corresponding item has to be filtered, non-zero means that corresponding item needs to stay).

```
#include <filter_layer.hpp>
```

Inheritance diagram for caffe::FilterLayer< Dtype >:



Collaboration diagram for caffe::FilterLayer< Dtype >:



Public Member Functions

- **FilterLayer** (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void Reshape (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.
- virtual const char ∗ type () const

Returns the layer type.
- virtual int MinBottomBlobs () const

  Returns the minimum number of bottom blobs required by the layer, or -1 if no minimum number is required.
- virtual int MinTopBlobs () const

  Returns the minimum number of top blobs required by the layer, or -1 if no minimum number is required.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Computes the error gradient w.r.t. the forwarded inputs.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Protected Attributes

- bool **first_reshape_**
- vector< int > **indices_to_forward_**

5.41.1 Detailed Description

template<typename Dtype>
class caffe::FilterLayer< Dtype >

Takes two+ Blobs, interprets last Blob as a selector and filter remaining Blobs accordingly with selector data (0 means that the corresponding item has to be filtered, non-zero means that corresponding item needs to stay).

5.41.2 Member Function Documentation

5.41.2.1 Backward_cpu()

```
template<typename Dtype >
void caffe::FilterLayer< Dtype >::Backward_cpu (
            const vector< Blob< Dtype > *> & top,
            const vector< bool > & propagate_down,
            const vector< Blob< Dtype > *> & bottom )  [protected], [virtual]
```

Computes the error gradient w.r.t. the forwarded inputs.

Parameters

top	output Blob vector (length 1+), providing the error gradient with respect to the outputs
propagate_down	see Layer::Backward.
bottom	input Blob vector (length 2+), into which the top error gradient is copied

Implements caffe::Layer< Dtype >.

5.41.2.2 Forward_cpu()

```
template<typename Dtype >
void caffe::FilterLayer< Dtype >::Forward_cpu (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [protected], [virtual]
```

Parameters

bottom	input Blob vector (length 2+) 1. $(N \times C \times H \times W)$ the inputs to be filtered x_1 2. ... 3. $(N \times C \times H \times W)$ the inputs to be filtered x_K 4. $(N \times 1 \times 1 \times 1)$ the selector blob
top	output Blob vector (length 1+) 1. $(S \times C \times H \times W)$ () the filtered output x_1 where S is the number of items that haven't been filtered $(S \times C \times H \times W)$ the filtered output x_K where S is the number of items that haven't been filtered

Implements caffe::Layer< Dtype >.

5.41.2.3 LayerSetUp()

```
template<typename Dtype >
void caffe::FilterLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

5.41.2.4 MinBottomBlobs()

```
template<typename Dtype >
virtual int caffe::FilterLayer< Dtype >::MinBottomBlobs ( ) const [inline], [virtual]
```

Returns the minimum number of bottom blobs required by the layer, or -1 if no minimum number is required.

This method should be overridden to return a non-negative value if your layer expects some minimum number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.41.2.5 MinTopBlobs()

```
template<typename Dtype >
virtual int caffe::FilterLayer< Dtype >::MinTopBlobs ( ) const [inline], [virtual]
```

Returns the minimum number of top blobs required by the layer, or -1 if no minimum number is required.

This method should be overridden to return a non-negative value if your layer expects some minimum number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.41.2.6 Reshape()

```
template<typename Dtype >
void caffe::FilterLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as

reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

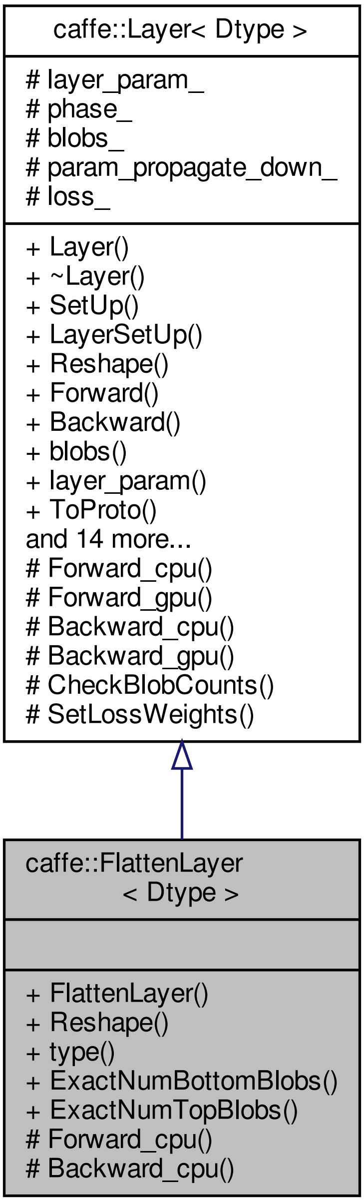
- include/caffe/layers/filter_layer.hpp
- src/caffe/layers/filter_layer.cpp

5.42 caffe::FlattenLayer< Dtype > Class Template Reference

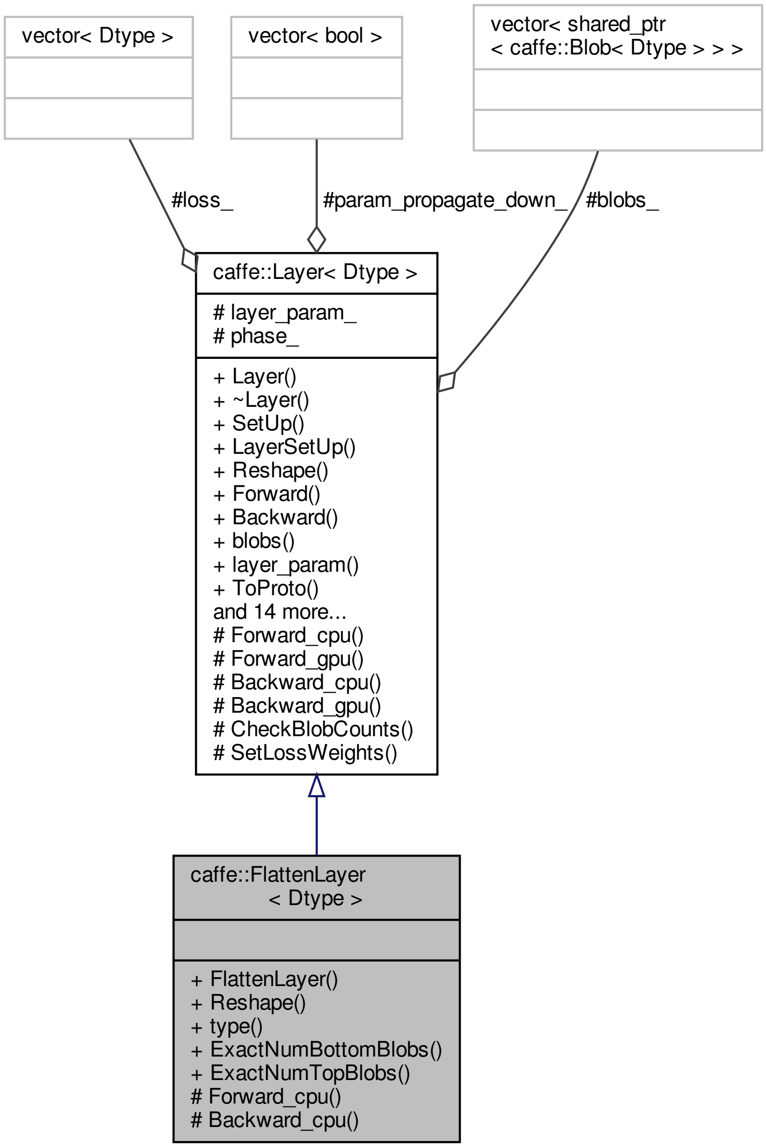
Reshapes the input Blob into flat vectors.

`#include <flatten_layer.hpp>`

Inheritance diagram for caffe::FlattenLayer< Dtype >:



Collaboration diagram for caffe::FlattenLayer< Dtype >:



Public Member Functions

- **FlattenLayer** (const LayerParameter &param)
- virtual void Reshape (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.
- virtual const char ∗ type () const

  Returns the layer type.
- virtual int ExactNumBottomBlobs () const

  Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.
- virtual int ExactNumTopBlobs () const

  Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Computes the error gradient w.r.t. the concatenate inputs.

Additional Inherited Members

5.42.1 Detailed Description

template<typename Dtype>
class caffe::FlattenLayer< Dtype >

Reshapes the input Blob into flat vectors.

Note: because this layer does not change the input values – merely the dimensions – it can simply copy the input. The copy happens "virtually" (thus taking effectively 0 real time) by setting, in Forward, the data pointer of the top Blob to that of the bottom Blob (see Blob::ShareData), and in Backward, the diff pointer of the bottom Blob to that of the top Blob (see Blob::ShareDiff).

5.42.2 Member Function Documentation

5.42.2.1 Backward_cpu()

```
template<typename Dtype >
void caffe::FlattenLayer< Dtype >::Backward_cpu (
            const vector< Blob< Dtype > *> & top,
            const vector< bool > & propagate_down,
            const vector< Blob< Dtype > *> & bottom )  [protected], [virtual]
```

Computes the error gradient w.r.t. the concatenate inputs.

Parameters

top	output Blob vector (length 1), providing the error gradient with respect to the outputs
propagate_down	see Layer::Backward.
bottom	input Blob vector (length K), into which the top error gradient is (virtually) copied

Implements caffe::Layer< Dtype >.

5.42.2.2 ExactNumBottomBlobs()

```
template<typename Dtype >
virtual int caffe::FlattenLayer< Dtype >::ExactNumBottomBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.42.2.3 ExactNumTopBlobs()

```
template<typename Dtype >
virtual int caffe::FlattenLayer< Dtype >::ExactNumTopBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.42.2.4 Forward_cpu()

```
template<typename Dtype >
void caffe::FlattenLayer< Dtype >::Forward_cpu (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [protected], [virtual]
```

Parameters

bottom	input Blob vector (length 2+) 1. $(N \times C \times H \times W)$ the inputs
top	output Blob vector (length 1) 1. $(N \times CHW \times 1 \times 1)$ the outputs – i.e., the (virtually) copied, flattened inputs

Implements caffe::Layer< Dtype >.

5.42.2.5 Reshape()

```
template<typename Dtype >
void caffe::FlattenLayer< Dtype >::Reshape (
```

```
const vector< Blob< Dtype > *> & bottom,
const vector< Blob< Dtype > *> & top )  [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

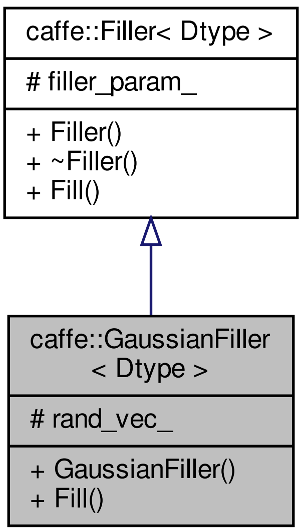
- include/caffe/layers/flatten_layer.hpp
- src/caffe/layers/flatten_layer.cpp

5.43 caffe::GaussianFiller< Dtype > Class Template Reference

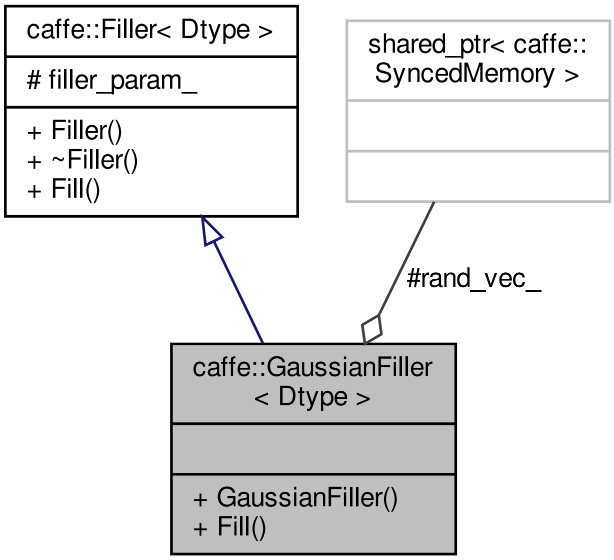
Fills a Blob with Gaussian-distributed values $x = a$.

```
#include <filler.hpp>
```

Inheritance diagram for caffe::GaussianFiller< Dtype >:



Collaboration diagram for caffe::GaussianFiller< Dtype >:



Public Member Functions

- **GaussianFiller** (const FillerParameter &param)
- virtual void **Fill** (Blob< Dtype > *blob)

Protected Attributes

- shared_ptr< SyncedMemory > **rand_vec_**

5.43.1 Detailed Description

template<typename Dtype>
class caffe::GaussianFiller< Dtype >

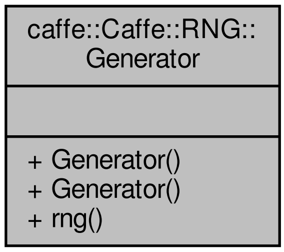
Fills a Blob with Gaussian-distributed values $x = a$.

The documentation for this class was generated from the following file:

- include/caffe/filler.hpp

5.44 caffe::Caffe::RNG::Generator Class Reference

Collaboration diagram for caffe::Caffe::RNG::Generator:



Public Member Functions

- **Generator** (unsigned int seed)
- caffe::rng_t ∗ **rng** ()

The documentation for this class was generated from the following file:

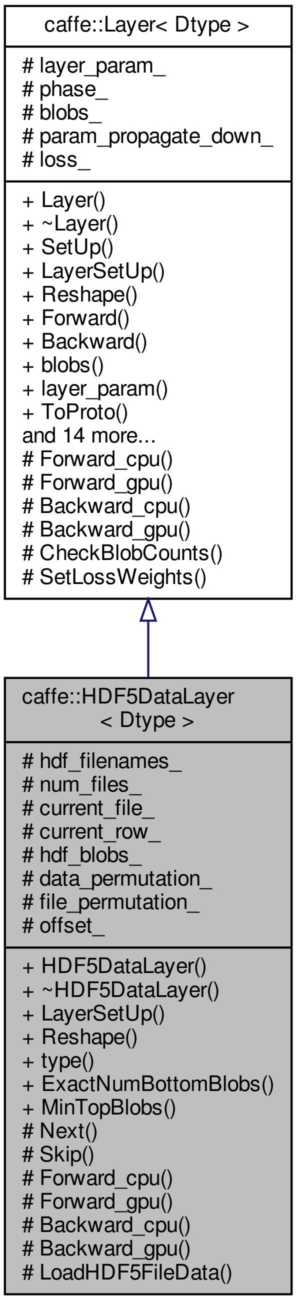
- src/caffe/common.cpp

5.45 caffe::HDF5DataLayer< Dtype > Class Template Reference

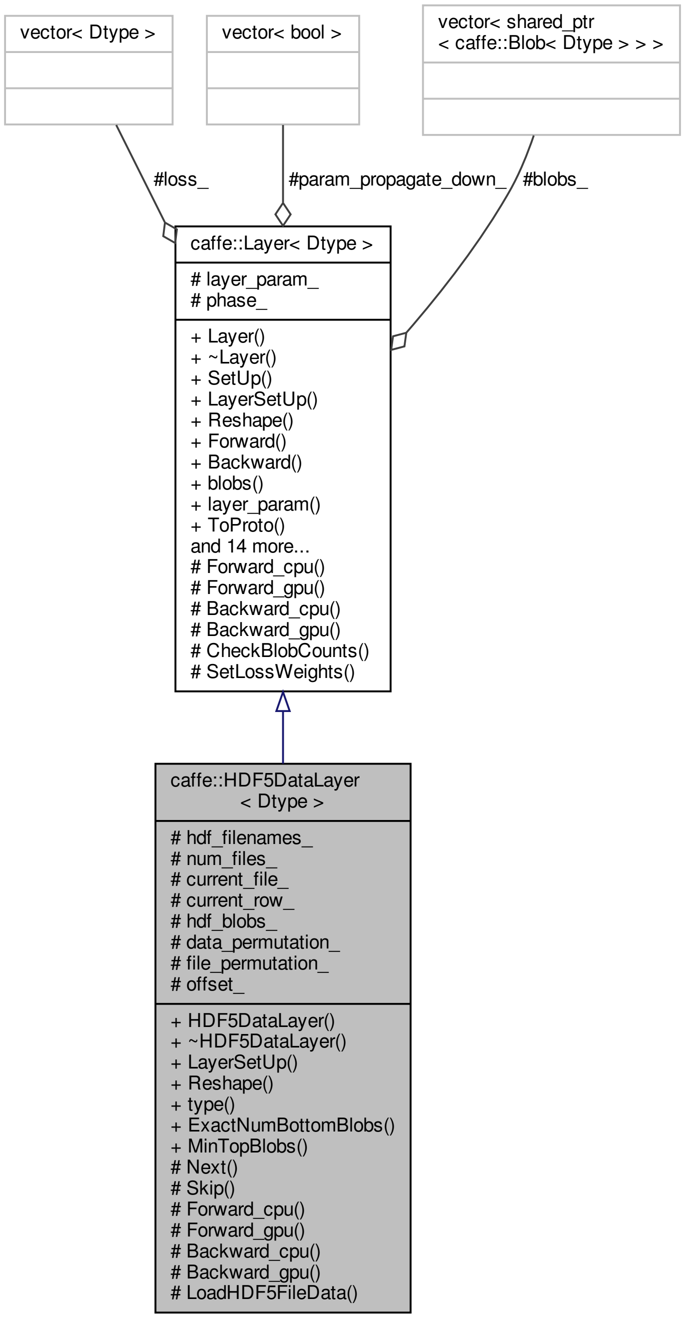
Provides data to the Net from HDF5 files.

```
#include <hdf5_data_layer.hpp>
```

Inheritance diagram for caffe::HDF5DataLayer< Dtype >:



Collaboration diagram for caffe::HDF5DataLayer< Dtype >:



Public Member Functions

- **HDF5DataLayer** (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

    Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void Reshape (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

    Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

- virtual const char * type () const

  Returns the layer type.
- virtual int ExactNumBottomBlobs () const

  Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.
- virtual int MinTopBlobs () const

  Returns the minimum number of top blobs required by the layer, or -1 if no minimum number is required.

Protected Member Functions

- void **Next** ()
- bool **Skip** ()
- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the CPU device, compute the layer output.
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the CPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.
- virtual void **LoadHDF5FileData** (const char *filename)

Protected Attributes

- std::vector< std::string > **hdf_filenames_**
- unsigned int **num_files_**
- unsigned int **current_file_**
- hsize_t **current_row_**
- std::vector< shared_ptr< Blob< Dtype > > > **hdf_blobs_**
- std::vector< unsigned int > **data_permutation_**
- std::vector< unsigned int > **file_permutation_**
- uint64_t **offset_**

5.45.1 Detailed Description

template<typename Dtype>
class caffe::HDF5DataLayer< Dtype >

Provides data to the Net from HDF5 files.

TODO(dox): thorough documentation for Forward and proto params.

5.45.2 Member Function Documentation

5.45.2.1 ExactNumBottomBlobs()

```
template<typename Dtype >
virtual int caffe::HDF5DataLayer< Dtype >::ExactNumBottomBlobs ( ) const [inline], [virtual]
```

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.45.2.2 LayerSetUp()

```
template<typename Dtype >
virtual void caffe::HDF5DataLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

5.45.2.3 MinTopBlobs()

```
template<typename Dtype >
virtual int caffe::HDF5DataLayer< Dtype >::MinTopBlobs ( ) const [inline], [virtual]
```

Returns the minimum number of top blobs required by the layer, or -1 if no minimum number is required.

This method should be overridden to return a non-negative value if your layer expects some minimum number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.45.2.4 Reshape()

```
template<typename Dtype >
virtual void caffe::HDF5DataLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [inline], [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following file:

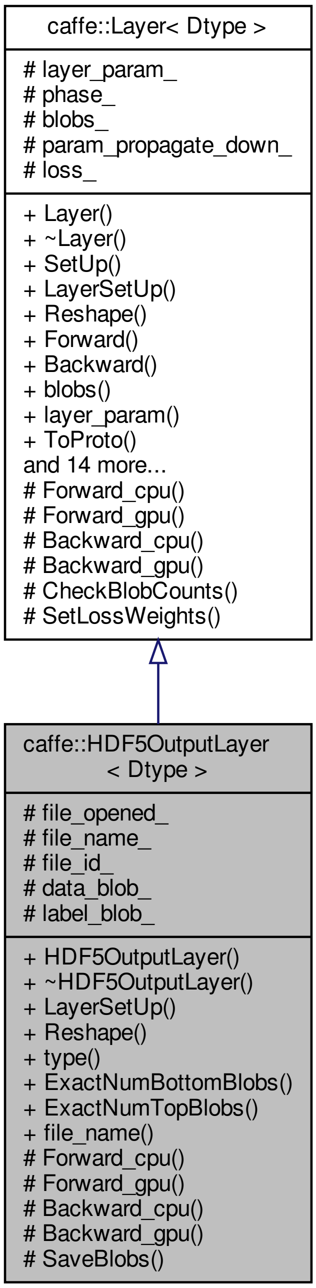
- include/caffe/layers/hdf5_data_layer.hpp

5.46 caffe::HDF5OutputLayer< Dtype > Class Template Reference

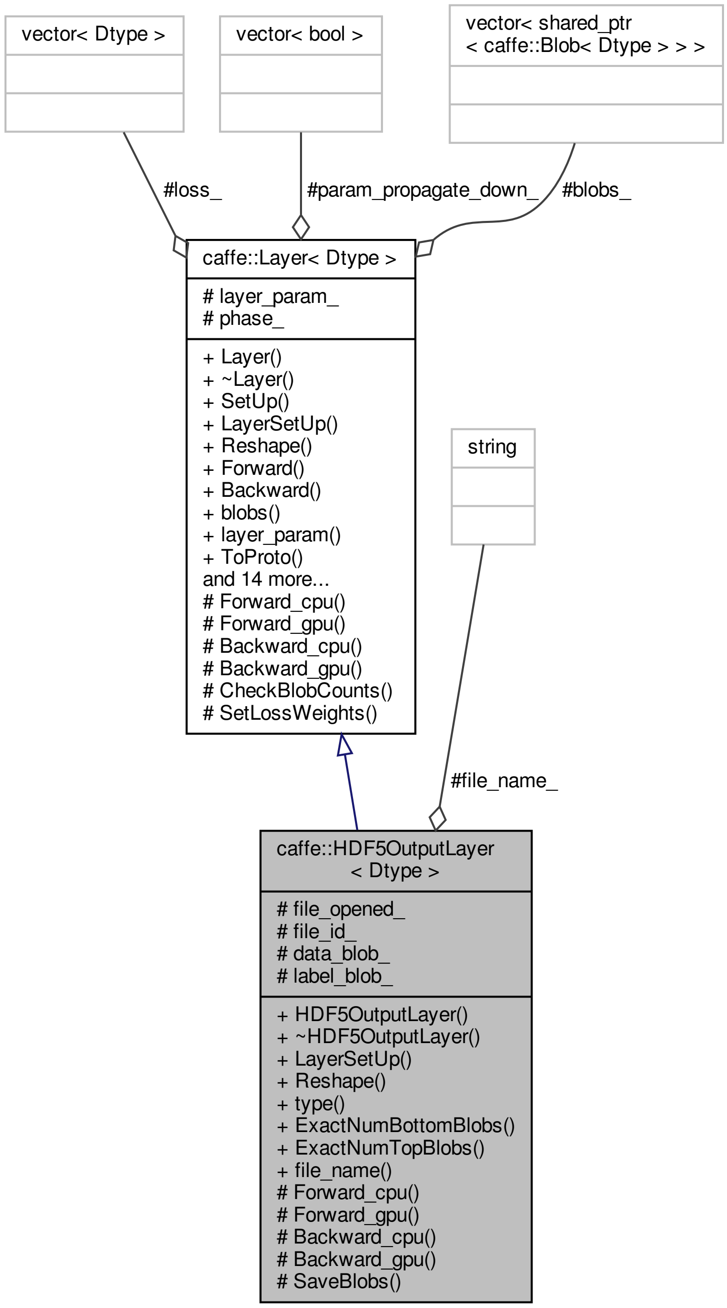
Write blobs to disk as HDF5 files.

```
#include <hdf5_output_layer.hpp>
```

Inheritance diagram for caffe::HDF5OutputLayer< Dtype >:



Collaboration diagram for caffe::HDF5OutputLayer< Dtype >:



Public Member Functions

- **HDF5OutputLayer** (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

    Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void Reshape (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

    Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

- virtual const char * type () const

  Returns the layer type.
- virtual int ExactNumBottomBlobs () const

  Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.
- virtual int ExactNumTopBlobs () const

  Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.
- std::string **file_name** () const

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the CPU device, compute the layer output.
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the CPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.
- virtual void **SaveBlobs** ()

Protected Attributes

- bool **file_opened_**
- std::string **file_name_**
- hid_t **file_id_**
- Blob< Dtype > **data_blob_**
- Blob< Dtype > **label_blob_**

5.46.1 Detailed Description

template<typename Dtype>
class caffe::HDF5OutputLayer< Dtype >

Write blobs to disk as HDF5 files.

TODO(dox): thorough documentation for Forward and proto params.

5.46.2 Member Function Documentation

5.46.2.1 ExactNumBottomBlobs()

```
template<typename Dtype >
virtual int caffe::HDF5OutputLayer< Dtype >::ExactNumBottomBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.46.2.2 ExactNumTopBlobs()

```
template<typename Dtype >
virtual int caffe::HDF5OutputLayer< Dtype >::ExactNumTopBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.46.2.3 LayerSetUp()

```
template<typename Dtype >
virtual void caffe::HDF5OutputLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

5.46.2.4 Reshape()

```
template<typename Dtype >
virtual void caffe::HDF5OutputLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [inline], [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following file:

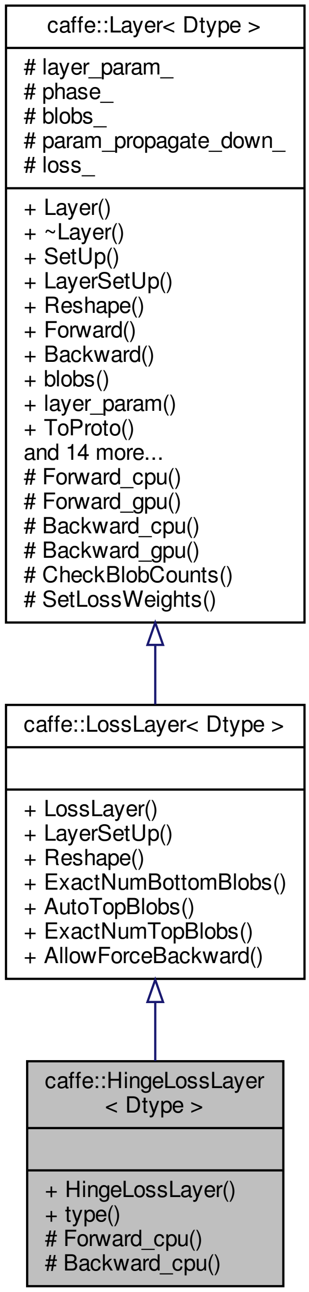
- include/caffe/layers/hdf5_output_layer.hpp

5.47 caffe::HingeLossLayer< Dtype > Class Template Reference

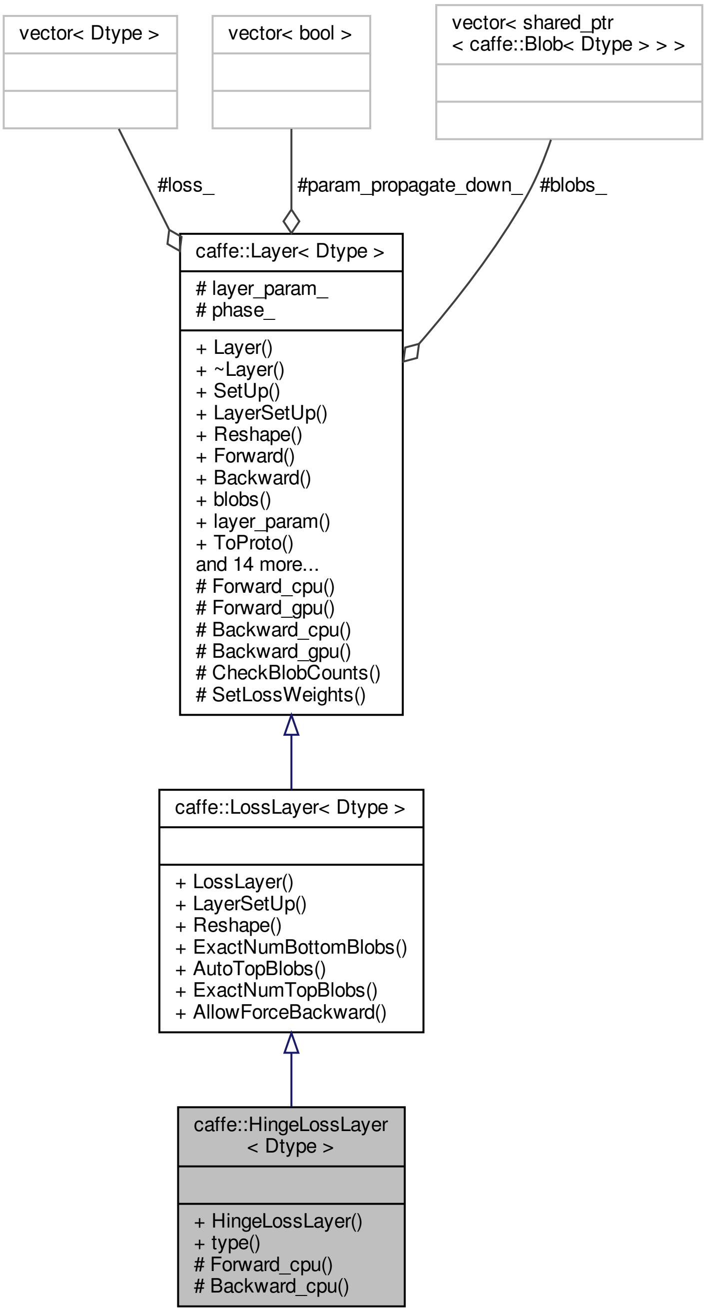
Computes the hinge loss for a one-of-many classification task.

```
#include <hinge_loss_layer.hpp>
```

Inheritance diagram for caffe::HingeLossLayer< Dtype >:



Collaboration diagram for caffe::HingeLossLayer< Dtype >:



Public Member Functions

- **HingeLossLayer** (const LayerParameter &param)
- virtual const char ∗ type () const

  Returns the layer type.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Computes the hinge loss for a one-of-many classification task.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Computes the hinge loss error gradient w.r.t. the predictions.

Additional Inherited Members

5.47.1 Detailed Description

template<typename Dtype>
class caffe::HingeLossLayer< Dtype >

Computes the hinge loss for a one-of-many classification task.

Parameters

bottom	input Blob vector (length 2) 1. $(N \times C \times H \times W)$ the predictions t, a Blob with values in $[-\infty, +\infty]$ indicating the predicted score for each of the $K = CHW$ classes. In an SVM, t is the result of taking the inner product $X^T W$ of the D-dimensional features $X \in \mathcal{R}^{D \times N}$ and the learned hyperplane parameters $W \in \mathcal{R}^{D \times K}$, so a Net with just an InnerProductLayer (with num_output = D) providing predictions to a HingeLossLayer and no other learnable parameters or losses is equivalent to an SVM. 2. $(N \times 1 \times 1 \times 1)$ the labels l, an integer-valued Blob with values $l_n \in [0, 1, 2, ..., K-1]$ indicating the correct class label among the K classes
top	output Blob vector (length 1) 1. $(1 \times 1 \times 1 \times 1)$ the computed hinge loss: $E = \frac{1}{N} \sum\limits_{n=1}^{N} \sum\limits_{k=1}^{K} [\max(0, 1 - \delta\{l_n = k\} t_{nk})]^p$, for the L^p norm (defaults to $p = 1$, the L1 norm; L2 norm, as in L2-SVM, is also available), and $\delta\{\mathrm{condition}\} = \left\{ \begin{array}{lr} 1 & \mbox{if condition} \\ -1 & \mbox{otherwise} \end{array} \right.$

In an SVM, $t \in \mathcal{R}^{N \times K}$ is the result of taking the inner product $X^T W$ of the features $X \in \mathcal{R}^{D \times N}$ and the learned hyperplane parameters $W \in \mathcal{R}^{D \times K}$. So, a Net with just an InnerProductLayer (with num_output = k) providing predictions to a HingeLossLayer is equivalent to an SVM (assuming it has no other learned outside the InnerProductLayer and no other losses outside the HingeLossLayer).

5.47.2 Member Function Documentation

5.47.2.1 Backward_cpu()

```
template<typename Dtype >
void caffe::HingeLossLayer< Dtype >::Backward_cpu (
            const vector< Blob< Dtype > *> & top,
            const vector< bool > & propagate_down,
            const vector< Blob< Dtype > *> & bottom ) [protected], [virtual]
```

Computes the hinge loss error gradient w.r.t. the predictions.

Gradients cannot be computed with respect to the label inputs (bottom[1]), so this method ignores bottom[1] and requires !propagate_down[1], crashing if propagate_down[1] is set.

Parameters

top	output Blob vector (length 1), providing the error gradient with respect to the outputs 1. $(1 \times 1 \times 1 \times 1)$ This Blob's diff will simply contain the loss_weight$*$ λ, as λ is the coefficient of this layer's output ℓ_i in the overall Net loss $E = \lambda_i \ell_i +$ other loss terms; hence $\frac{\partial E}{\partial \ell_i} = \lambda_i$. ($*$Assuming that this top Blob is not used as a bottom (input) by any other layer of the Net.)
propagate_down	see Layer::Backward. propagate_down[1] must be false as we can't compute gradients with respect to the labels.
bottom	input Blob vector (length 2) 1. $(N \times C \times H \times W)$ the predictions t; Backward computes diff $\frac{\partial E}{\partial t}$ 2. $(N \times 1 \times 1 \times 1)$ the labels – ignored as we can't compute their error gradients

Implements caffe::Layer< Dtype >.

5.47.2.2 Forward_cpu()

```
template<typename Dtype >
void caffe::HingeLossLayer< Dtype >::Forward_cpu (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [protected], [virtual]
```

Computes the hinge loss for a one-of-many classification task.

Parameters

bottom	input Blob vector (length 2) 1. $(N \times C \times H \times W)$ the predictions t, a Blob with values in $[-\infty, +\infty]$ indicating the predicted score for each of the $K = CHW$ classes. In an SVM, t is the result of taking the inner product $X^T W$ of the D-dimensional features $X \in \mathcal{R}^{D \times N}$ and the learned hyperplane parameters $W \in \mathcal{R}^{D \times K}$, so a Net with just an InnerProductLayer (with num_output = D) providing predictions to a HingeLossLayer and no other learnable parameters or losses is equivalent to an SVM. 2. $(N \times 1 \times 1 \times 1)$ the labels l, an integer-valued Blob with values $l_n \in [0, 1, 2, ..., K-1]$ indicating the correct class label among the K classes
top	output Blob vector (length 1) 1. $(1 \times 1 \times 1 \times 1)$ the computed hinge loss: $E = \frac{1}{N} \sum\limits_{n=1}^{N} \sum\limits_{k=1}^{K} [\max(0, 1 - \delta\{l_n = k\} t_{nk})]^p$, for the L^p norm (defaults to $p = 1$, the L1 norm; L2 norm, as in L2-SVM, is also available), and $\delta\{\mathrm{condition}\} = \begin{cases} 1 & \text{if condition} \\ -1 & \text{otherwise} \end{cases}$

In an SVM, $t \in \mathcal{R}^{N \times K}$ is the result of taking the inner product $X^T W$ of the features $X \in \mathcal{R}^{D \times N}$ and the learned hyperplane parameters $W \in \mathcal{R}^{D \times K}$. So, a Net with just an InnerProductLayer (with num_output = k) providing predictions to a HingeLossLayer is equivalent to an SVM (assuming it has no other learned outside the InnerProductLayer and no other losses outside the HingeLossLayer).

Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

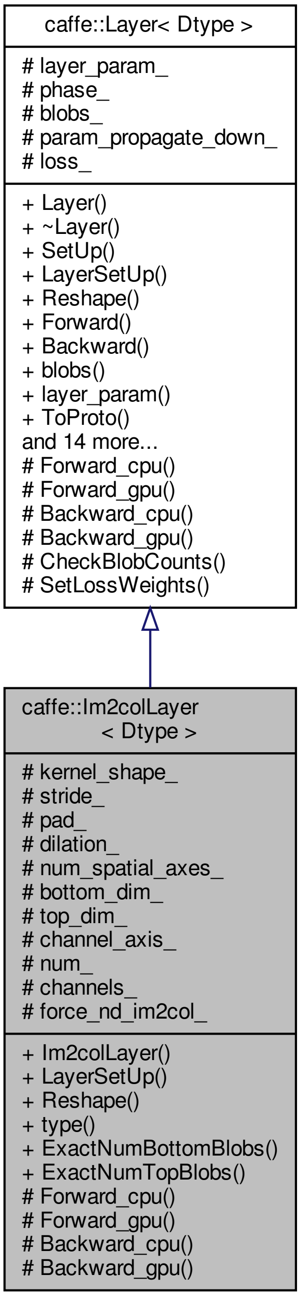
- include/caffe/layers/hinge_loss_layer.hpp
- src/caffe/layers/hinge_loss_layer.cpp

5.48 caffe::Im2colLayer< Dtype > Class Template Reference

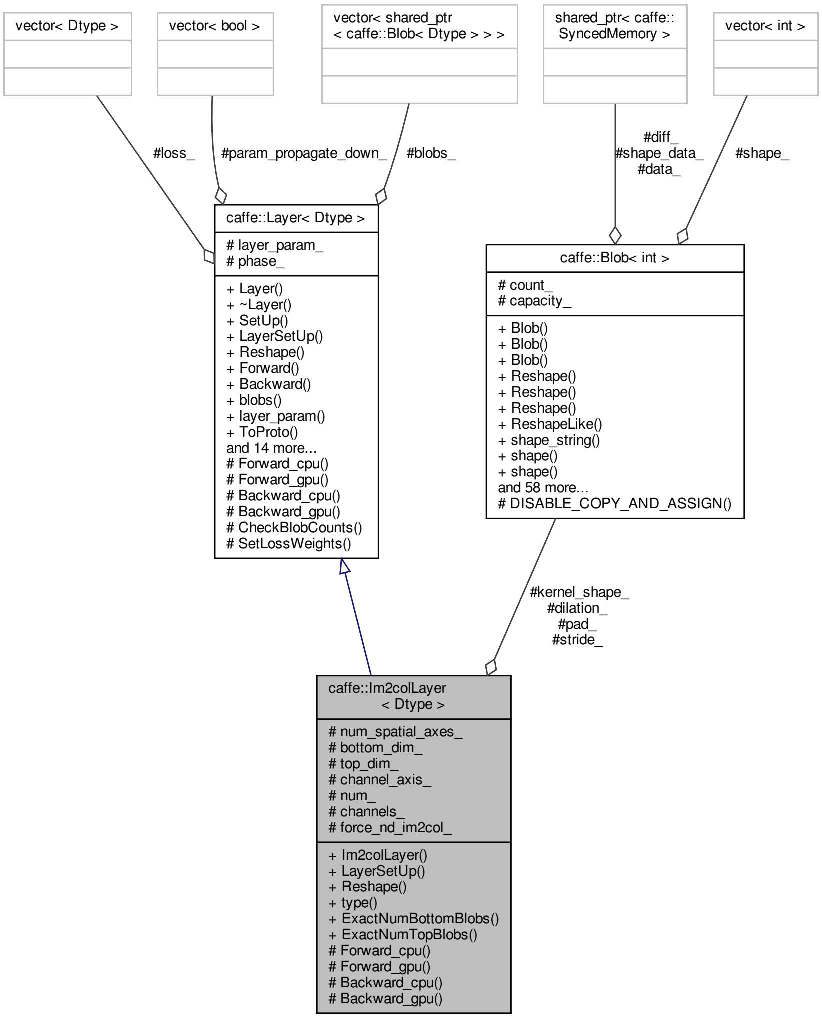
A helper for image operations that rearranges image regions into column vectors. Used by ConvolutionLayer to perform convolution by matrix multiplication.

```
#include <im2col_layer.hpp>
```

Inheritance diagram for caffe::Im2colLayer< Dtype >:



Collaboration diagram for caffe::Im2colLayer< Dtype >:



Public Member Functions

- **Im2colLayer** (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void Reshape (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.
- virtual const char * type () const

  Returns the layer type.
- virtual int ExactNumBottomBlobs () const

  Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.
- virtual int ExactNumTopBlobs () const

  Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the CPU device, compute the layer output.
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the CPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Protected Attributes

- Blob< int > kernel_shape_

  The spatial dimensions of a filter kernel.
- Blob< int > stride_

  The spatial dimensions of the stride.
- Blob< int > pad_

  The spatial dimensions of the padding.
- Blob< int > dilation_

  The spatial dimensions of the dilation.
- int **num_spatial_axes_**
- int **bottom_dim_**
- int **top_dim_**
- int **channel_axis_**
- int **num_**
- int **channels_**
- bool **force_nd_im2col_**

5.48.1 Detailed Description

template<typename Dtype>
class caffe::Im2colLayer< Dtype >

A helper for image operations that rearranges image regions into column vectors. Used by ConvolutionLayer to perform convolution by matrix multiplication.

TODO(dox): thorough documentation for Forward, Backward, and proto params.

5.48.2 Member Function Documentation

5.48.2.1 ExactNumBottomBlobs()

```
template<typename Dtype >
virtual int caffe::Im2colLayer< Dtype >::ExactNumBottomBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.48.2.2 ExactNumTopBlobs()

```
template<typename Dtype >
virtual int caffe::Im2colLayer< Dtype >::ExactNumTopBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.48.2.3 LayerSetUp()

```
template<typename Dtype >
void caffe::Im2colLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

5.48.2.4 Reshape()

```
template<typename Dtype >
void caffe::Im2colLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

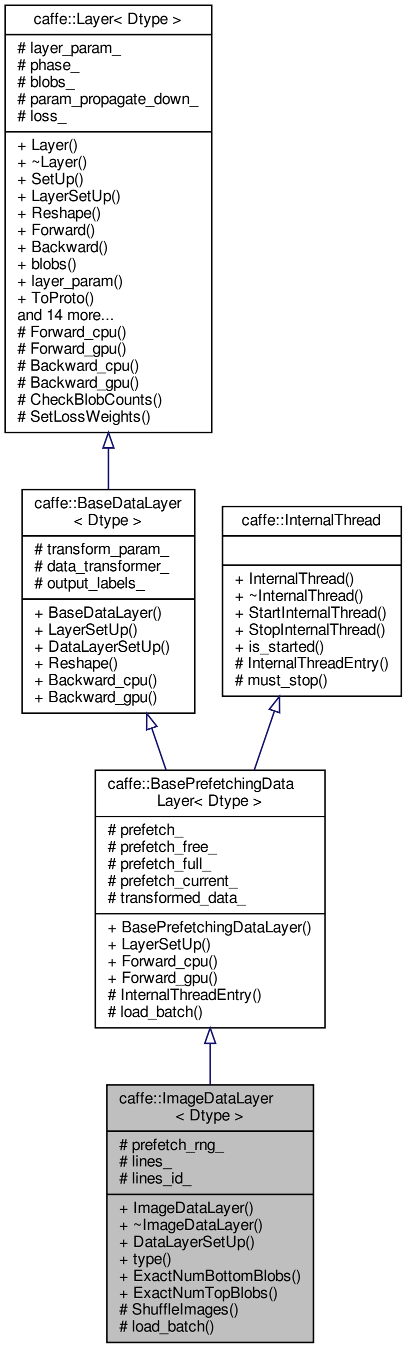
- include/caffe/layers/im2col_layer.hpp
- src/caffe/layers/im2col_layer.cpp

5.49 caffe::ImageDataLayer< Dtype > Class Template Reference

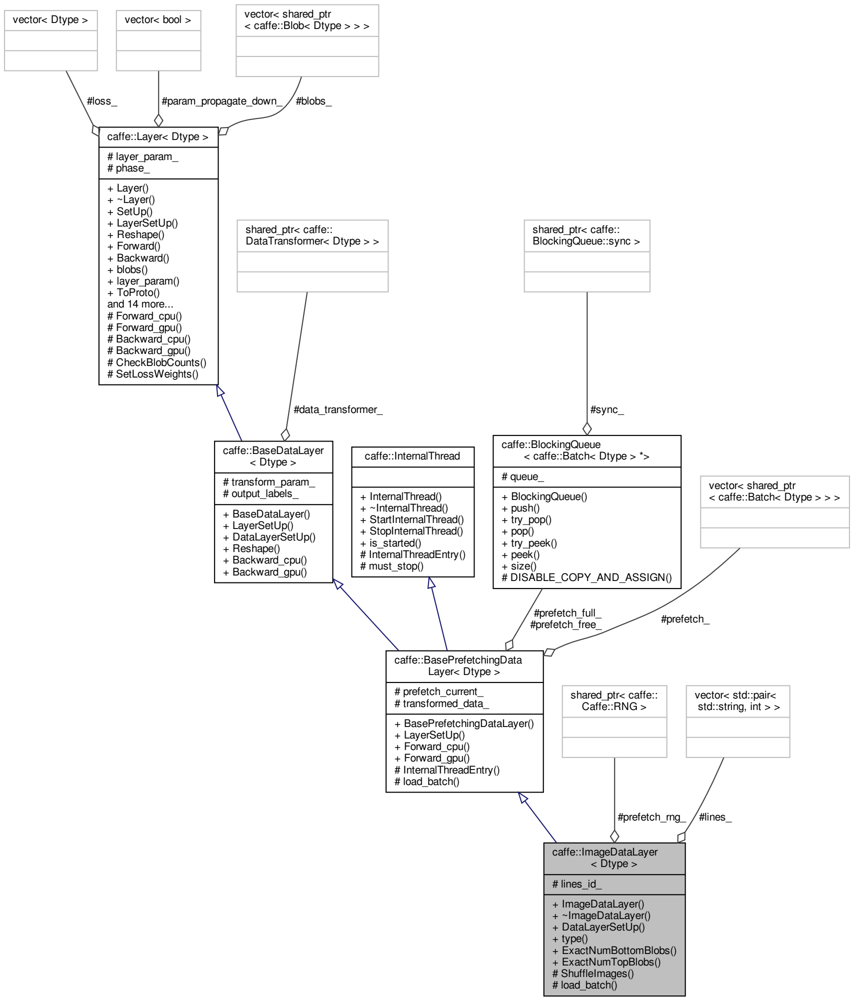
Provides data to the Net from image files.

```
#include <image_data_layer.hpp>
```

Inheritance diagram for caffe::ImageDataLayer< Dtype >:



Collaboration diagram for caffe::ImageDataLayer< Dtype >:



Public Member Functions

- **ImageDataLayer** (const LayerParameter &param)
- virtual void **DataLayerSetUp** (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)
- virtual const char ∗ type () const

    Returns the layer type.
- virtual int ExactNumBottomBlobs () const

    Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.
- virtual int ExactNumTopBlobs () const

    Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

Protected Member Functions

- virtual void **ShuffleImages** ()
- virtual void **load_batch** (Batch< Dtype > ∗batch)

Protected Attributes

- shared_ptr< Caffe::RNG > **prefetch_rng_**
- vector< std::pair< std::string, int > > **lines_**
- int **lines_id_**

5.49.1 Detailed Description

template<typename Dtype>
class caffe::ImageDataLayer< Dtype >

Provides data to the Net from image files.

TODO(dox): thorough documentation for Forward and proto params.

5.49.2 Member Function Documentation

5.49.2.1 ExactNumBottomBlobs()

```
template<typename Dtype >
virtual int caffe::ImageDataLayer< Dtype >::ExactNumBottomBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.49.2.2 ExactNumTopBlobs()

```
template<typename Dtype >
virtual int caffe::ImageDataLayer< Dtype >::ExactNumTopBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

The documentation for this class was generated from the following file:

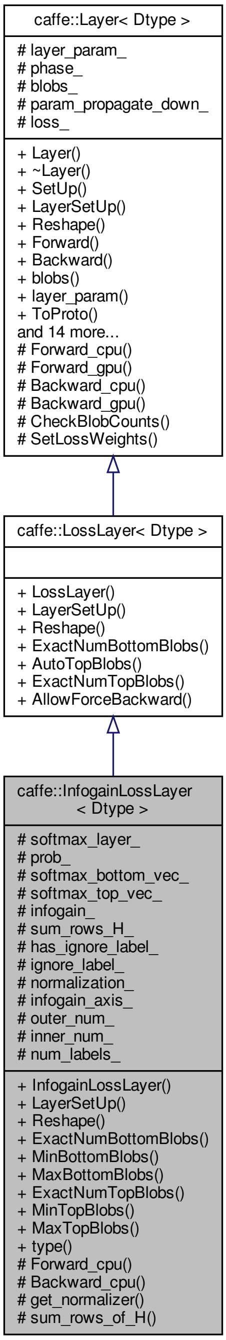
- include/caffe/layers/image_data_layer.hpp

5.50 caffe::InfogainLossLayer< Dtype > Class Template Reference

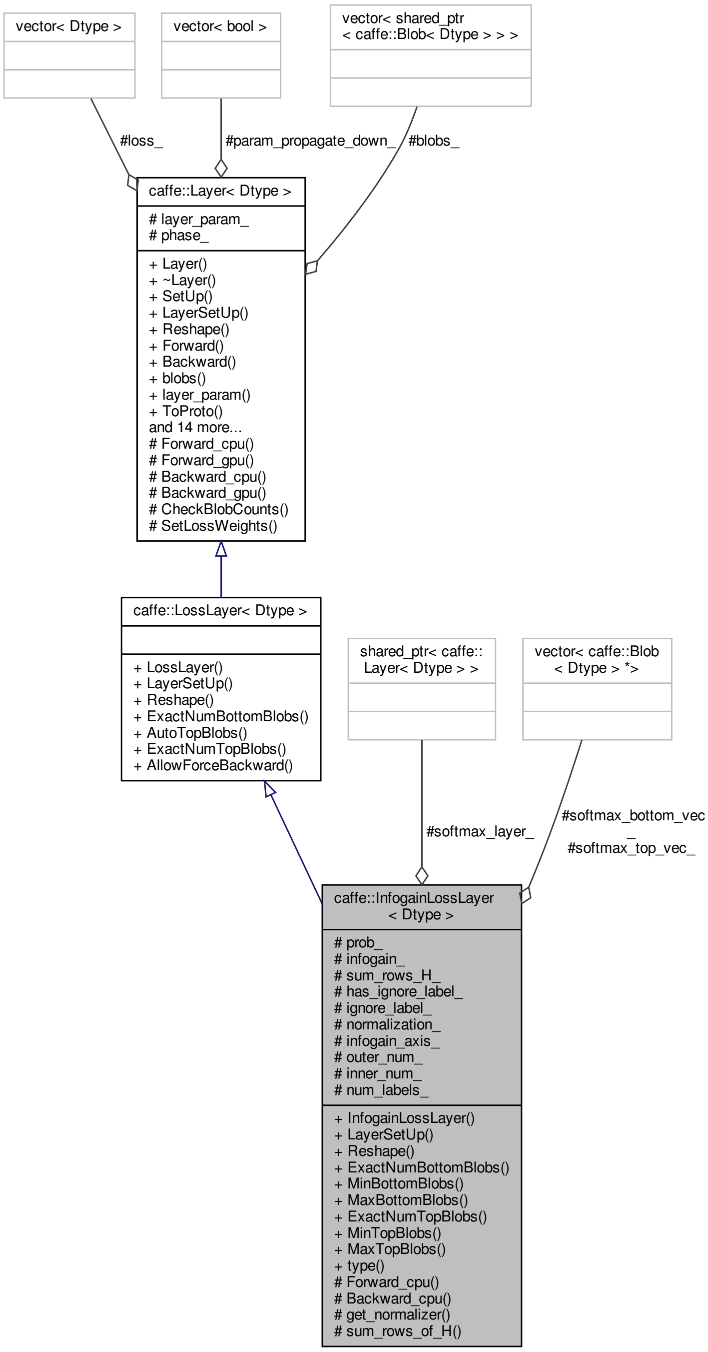
A generalization of SoftmaxWithLossLayer that takes an "information gain" (infogain) matrix specifying the "value" of all label pairs.

```
#include <infogain_loss_layer.hpp>
```

Inheritance diagram for caffe::InfogainLossLayer< Dtype >:



Collaboration diagram for caffe::InfogainLossLayer< Dtype >:



Public Member Functions

- **InfogainLossLayer** (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

    Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void Reshape (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

    Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

- virtual int ExactNumBottomBlobs () const

  Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.
- virtual int MinBottomBlobs () const

  Returns the minimum number of bottom blobs required by the layer, or -1 if no minimum number is required.
- virtual int MaxBottomBlobs () const

  Returns the maximum number of bottom blobs required by the layer, or -1 if no maximum number is required.
- virtual int ExactNumTopBlobs () const

  Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.
- virtual int MinTopBlobs () const

  Returns the minimum number of top blobs required by the layer, or -1 if no minimum number is required.
- virtual int MaxTopBlobs () const

  Returns the maximum number of top blobs required by the layer, or -1 if no maximum number is required.
- virtual const char * type () const

  Returns the layer type.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  A generalization of SoftmaxWithLossLayer that takes an "information gain" (infogain) matrix specifying the "value" of all label pairs.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Computes the infogain loss error gradient w.r.t. the predictions.
- virtual Dtype get_normalizer (LossParameter_NormalizationMode normalization_mode, int valid_count)
- virtual void sum_rows_of_H (const Blob< Dtype > *H)

  fill sum_rows_H_ according to matrix H

Protected Attributes

- shared_ptr< Layer< Dtype > > softmax_layer_

  The internal SoftmaxLayer used to map predictions to a distribution.
- Blob< Dtype > prob_

  prob stores the output probability predictions from the SoftmaxLayer.
- vector< Blob< Dtype > * > softmax_bottom_vec_

  bottom vector holder used in call to the underlying SoftmaxLayer::Forward
- vector< Blob< Dtype > * > softmax_top_vec_

  top vector holder used in call to the underlying SoftmaxLayer::Forward
- Blob< Dtype > **infogain_**
- Blob< Dtype > **sum_rows_H_**
- bool has_ignore_label_

  Whether to ignore instances with a certain label.
- int ignore_label_

  The label indicating that an instance should be ignored.
- LossParameter_NormalizationMode normalization_

  How to normalize the output loss.
- int **infogain_axis_**
- int **outer_num_**
- int **inner_num_**
- int **num_labels_**

5.50.1 Detailed Description

template<typename Dtype>
class caffe::InfogainLossLayer< Dtype >

A generalization of SoftmaxWithLossLayer that takes an "information gain" (infogain) matrix specifying the "value" of all label pairs.

Equivalent to the SoftmaxWithLossLayer if the infogain matrix is the identity.

Parameters

bottom	input Blob vector (length 2-3) 1. $(N \times C \times H \times W)$ the predictions x, a Blob with values in $[-\infty, +\infty]$ indicating the predicted score for each of the $K = CHW$ classes. This layer maps these scores to a probability distribution over classes using the softmax function $\hat{p}_{nk} = \exp(x_{nk}) / \left[\sum_{k'} \exp(x_{nk'})\right]$ (see SoftmaxLayer). 2. $(N \times 1 \times 1 \times 1)$ the labels l, an integer-valued Blob with values $l_n \in [0, 1, 2, ..., K-1]$ indicating the correct class label among the K classes 3. $(1 \times 1 \times K \times K)$ (**optional**) the infogain matrix H. This must be provided as the third bottom blob input if not provided as the infogain_mat in the InfogainLossParameter. If $H = I$, this layer is equivalent to the SoftmaxWithLossLayer.
top	output Blob vector (length 1) 1. $(1 \times 1 \times 1 \times 1)$ the computed infogain multinomial logistic loss: $E = \frac{-1}{N} \sum\limits_{n=1}^{N} H_{l_n} \log(\hat{p}_n) = \frac{-1}{N} \sum\limits_{n=1}^{N} \sum\limits_{k=1}^{K} H_{l_n,k} \log(\hat{p}_{n,k})$, where H_{l_n} denotes row l_n of H.

5.50.2 Member Function Documentation

5.50.2.1 Backward_cpu()

```
template<typename Dtype >
void caffe::InfogainLossLayer< Dtype >::Backward_cpu (
            const vector< Blob< Dtype > *> & top,
            const vector< bool > & propagate_down,
            const vector< Blob< Dtype > *> & bottom ) [protected], [virtual]
```

Computes the infogain loss error gradient w.r.t. the predictions.

Gradients cannot be computed with respect to the label inputs (bottom[1]), so this method ignores bottom[1] and requires !propagate_down[1], crashing if propagate_down[1] is set. (The same applies to the infogain matrix, if provided as bottom[2] rather than in the layer_param.)

Parameters

top	output Blob vector (length 1), providing the error gradient with respect to the outputs 1. $(1 \times 1 \times 1 \times 1)$ This Blob's diff will simply contain the loss_weight$*$ λ, as λ is the coefficient of this layer's output ℓ_i in the overall Net loss $E = \lambda_i \ell_i +$ other loss terms; hence $\frac{\partial E}{\partial \ell_i} = \lambda_i$. ($*$Assuming that this top Blob is not used as a bottom (input) by any other layer of the Net.)
propagate_down	see Layer::Backward. propagate_down[1] must be false as we can't compute gradients with respect to the labels (similarly for propagate_down[2] and the infogain matrix, if provided as bottom[2])
bottom	input Blob vector (length 2-3) 1. $(N \times C \times H \times W)$ the predictions x; Backward computes diff $\frac{\partial E}{\partial x}$ 2. $(N \times 1 \times 1 \times 1)$ the labels – ignored as we can't compute their error gradients 3. $(1 \times 1 \times K \times K)$ (**optional**) the information gain matrix – ignored as its error gradient computation is not implemented.

Implements caffe::Layer< Dtype >.

5.50.2.2 ExactNumBottomBlobs()

```
template<typename Dtype >
virtual int caffe::InfogainLossLayer< Dtype >::ExactNumBottomBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of bottom blobs.

Reimplemented from caffe::LossLayer< Dtype >.

5.50.2.3 ExactNumTopBlobs()

```
template<typename Dtype >
virtual int caffe::InfogainLossLayer< Dtype >::ExactNumTopBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of top blobs.

Reimplemented from caffe::LossLayer< Dtype >.

5.50.2.4 Forward_cpu()

```
template<typename Dtype >
void caffe::InfogainLossLayer< Dtype >::Forward_cpu (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [protected], [virtual]
```

A generalization of SoftmaxWithLossLayer that takes an "information gain" (infogain) matrix specifying the "value" of all label pairs.

Equivalent to the SoftmaxWithLossLayer if the infogain matrix is the identity.

Parameters

bottom	input Blob vector (length 2-3) 1. $(N \times C \times H \times W)$ the predictions x, a Blob with values in $[-\infty, +\infty]$ indicating the predicted score for each of the $K = CHW$ classes. This layer maps these scores to a probability distribution over classes using the softmax function $\hat{p}_{nk} = \exp(x_{nk}) / \left[\sum_{k'} \exp(x_{nk'})\right]$ (see SoftmaxLayer). 2. $(N \times 1 \times 1 \times 1)$ the labels l, an integer-valued Blob with values $l_n \in [0, 1, 2, ..., K-1]$ indicating the correct class label among the K classes 3. $(1 \times 1 \times K \times K)$ (**optional**) the infogain matrix H. This must be provided as the third bottom blob input if not provided as the infogain_mat in the InfogainLossParameter. If $H = I$, this layer is equivalent to the SoftmaxWithLossLayer.
top	output Blob vector (length 1) 1. $(1 \times 1 \times 1 \times 1)$ the computed infogain multinomial logistic loss: $E = \frac{-1}{N} \sum\limits_{n=1}^{N} H_{l_n} \log(\hat{p}_n) = \frac{-1}{N} \sum\limits_{n=1}^{N} \sum\limits_{k=1}^{K} H_{l_n,k} \log(\hat{p}_{n,k})$, where H_{l_n} denotes row l_n of H.

Implements caffe::Layer< Dtype >.

5.50.2.5 get_normalizer()

```
template<typename Dtype >
Dtype caffe::InfogainLossLayer< Dtype >::get_normalizer (
            LossParameter_NormalizationMode normalization_mode,
            int valid_count ) [protected], [virtual]
```

Read the normalization mode parameter and compute the normalizer based on the blob size. If normalization_↩ mode is VALID, the count of valid outputs will be read from valid_count, unless it is -1 in which case all outputs are assumed to be valid.

5.50.2.6 LayerSetUp()

```
template<typename Dtype >
void caffe::InfogainLossLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::LossLayer< Dtype >.

5.50.2.7 MaxBottomBlobs()

```
template<typename Dtype >
virtual int caffe::InfogainLossLayer< Dtype >::MaxBottomBlobs ( ) const  [inline], [virtual]
```

Returns the maximum number of bottom blobs required by the layer, or -1 if no maximum number is required.

This method should be overridden to return a non-negative value if your layer expects some maximum number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.50.2.8 MaxTopBlobs()

```
template<typename Dtype >
virtual int caffe::InfogainLossLayer< Dtype >::MaxTopBlobs ( ) const  [inline], [virtual]
```

Returns the maximum number of top blobs required by the layer, or -1 if no maximum number is required.

This method should be overridden to return a non-negative value if your layer expects some maximum number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.50.2.9 MinBottomBlobs()

```
template<typename Dtype >
virtual int caffe::InfogainLossLayer< Dtype >::MinBottomBlobs ( ) const  [inline], [virtual]
```

Returns the minimum number of bottom blobs required by the layer, or -1 if no minimum number is required.

This method should be overridden to return a non-negative value if your layer expects some minimum number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.50.2.10 MinTopBlobs()

```
template<typename Dtype >
virtual int caffe::InfogainLossLayer< Dtype >::MinTopBlobs ( ) const  [inline], [virtual]
```

Returns the minimum number of top blobs required by the layer, or -1 if no minimum number is required.

This method should be overridden to return a non-negative value if your layer expects some minimum number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.50.2.11 Reshape()

```
template<typename Dtype >
void caffe::InfogainLossLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Reimplemented from caffe::LossLayer< Dtype >.

The documentation for this class was generated from the following files:

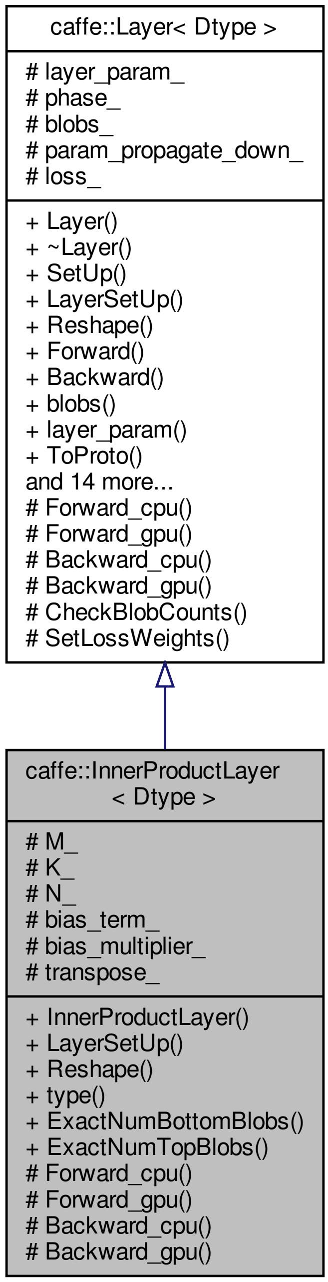
- include/caffe/layers/infogain_loss_layer.hpp
- src/caffe/layers/infogain_loss_layer.cpp

5.51 caffe::InnerProductLayer< Dtype > Class Template Reference

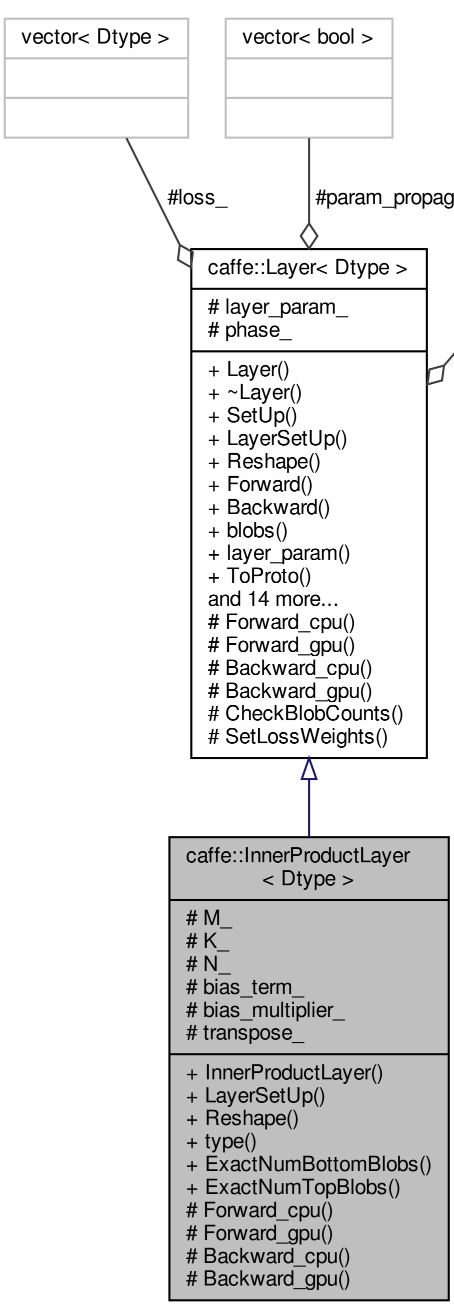
Also known as a "fully-connected" layer, computes an inner product with a set of learned weights, and (optionally) adds biases.

```
#include <inner_product_layer.hpp>
```

Inheritance diagram for caffe::InnerProductLayer< Dtype >:



Collaboration diagram for caffe::InnerProductLayer< Dtype >:



Public Member Functions

- **InnerProductLayer** (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void Reshape (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

- virtual const char $*$ type () const

  Returns the layer type.
- virtual int ExactNumBottomBlobs () const

  Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.
- virtual int ExactNumTopBlobs () const

  Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the CPU device, compute the layer output.
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the CPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Protected Attributes

- int **M_**
- int **K_**
- int **N_**
- bool **bias_term_**
- Blob< Dtype > **bias_multiplier_**
- bool transpose_

  if true, assume transposed weights

5.51.1 Detailed Description

template<typename Dtype>
class caffe::InnerProductLayer< Dtype >

Also known as a "fully-connected" layer, computes an inner product with a set of learned weights, and (optionally) adds biases.

TODO(dox): thorough documentation for Forward, Backward, and proto params.

5.51.2 Member Function Documentation

5.51.2.1 ExactNumBottomBlobs()

```
template<typename Dtype >
virtual int caffe::InnerProductLayer< Dtype >::ExactNumBottomBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.51.2.2 ExactNumTopBlobs()

```
template<typename Dtype >
virtual int caffe::InnerProductLayer< Dtype >::ExactNumTopBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.51.2.3 LayerSetUp()

```
template<typename Dtype >
void caffe::InnerProductLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

5.51.2.4 Reshape()

```
template<typename Dtype >
void caffe::InnerProductLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

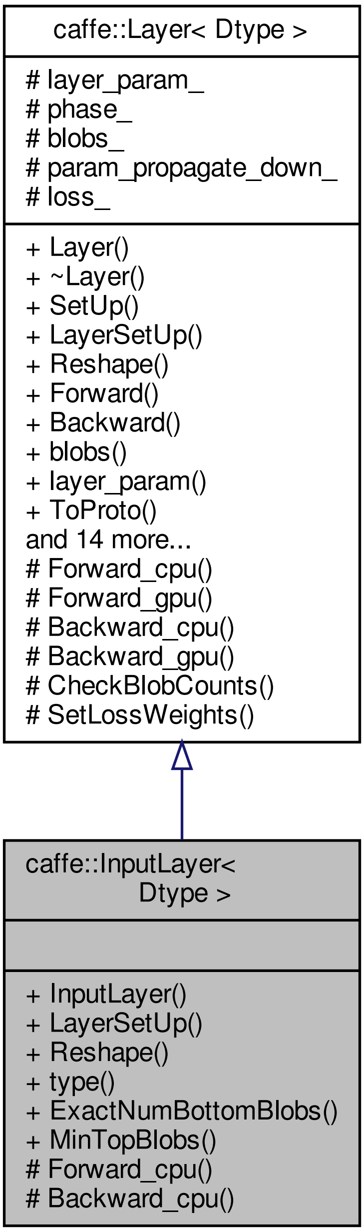
- include/caffe/layers/inner_product_layer.hpp
- src/caffe/layers/inner_product_layer.cpp

5.52 caffe::InputLayer< Dtype > Class Template Reference

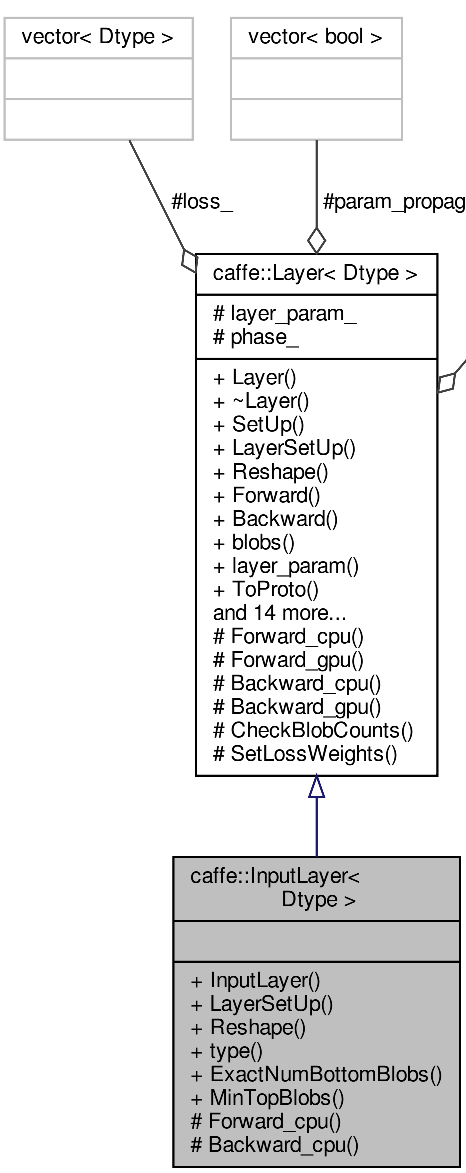
Provides data to the Net by assigning tops directly.

```
#include <input_layer.hpp>
```

Inheritance diagram for caffe::InputLayer< Dtype >:



Collaboration diagram for caffe::InputLayer< Dtype >:



Public Member Functions

- **InputLayer** (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void Reshape (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.
- virtual const char ∗ type () const

  Returns the layer type.
- virtual int ExactNumBottomBlobs () const

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

- virtual int MinTopBlobs () const

  Returns the minimum number of top blobs required by the layer, or -1 if no minimum number is required.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Using the CPU device, compute the layer output.
- virtual void Backward_cpu (const vector< Blob< Dtype > ∗> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > ∗> &bottom)

  Using the CPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true.

Additional Inherited Members

5.52.1 Detailed Description

template<typename Dtype>
class caffe::InputLayer< Dtype >

Provides data to the Net by assigning tops directly.

This data layer is a container that merely holds the data assigned to it; forward, backward, and reshape are all no-ops.

5.52.2 Member Function Documentation

5.52.2.1 ExactNumBottomBlobs()

```
template<typename Dtype >
virtual int caffe::InputLayer< Dtype >::ExactNumBottomBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.52.2.2 LayerSetUp()

```
template<typename Dtype >
void caffe::InputLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

5.52.2.3 MinTopBlobs()

```
template<typename Dtype >
virtual int caffe::InputLayer< Dtype >::MinTopBlobs ( ) const [inline], [virtual]
```

Returns the minimum number of top blobs required by the layer, or -1 if no minimum number is required.

This method should be overridden to return a non-negative value if your layer expects some minimum number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.52.2.4 Reshape()

```
template<typename Dtype >
virtual void caffe::InputLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [inline], [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Implements caffe::Layer< Dtype >.

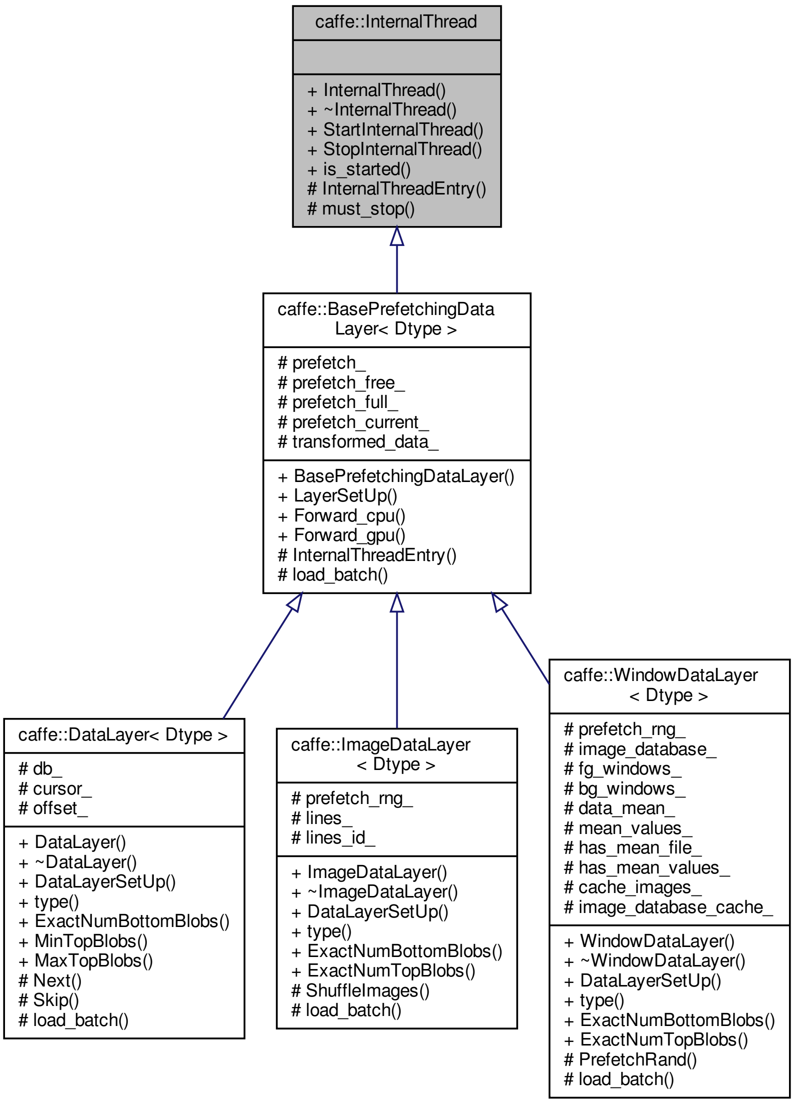
The documentation for this class was generated from the following files:

- include/caffe/layers/input_layer.hpp
- src/caffe/layers/input_layer.cpp

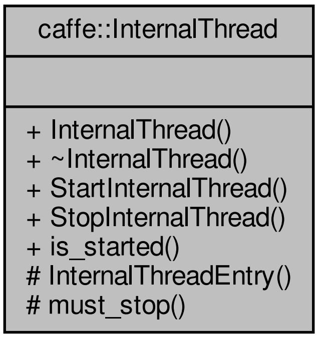
5.53 caffe::InternalThread Class Reference

`#include <internal_thread.hpp>`

Inheritance diagram for caffe::InternalThread:



Collaboration diagram for caffe::InternalThread:



Public Member Functions

- void StartInternalThread ()
- void StopInternalThread ()
- bool **is_started** () const

Protected Member Functions

- virtual void **InternalThreadEntry** ()
- bool **must_stop** ()

5.53.1 Detailed Description

Virtual class encapsulate boost::thread for use in base class The child class will acquire the ability to run a single thread, by reimplementing the virtual function InternalThreadEntry.

5.53.2 Member Function Documentation

5.53.2.1 StartInternalThread()

`void caffe::InternalThread::StartInternalThread ( )`

Caffe's thread local state will be initialized using the current thread values, e.g. device id, solver index etc. The random seed is initialized using caffe_rng_rand.

5.53.2.2 StopInternalThread()

`void caffe::InternalThread::StopInternalThread ( )`

Will not return until the internal thread has exited.

The documentation for this class was generated from the following files:

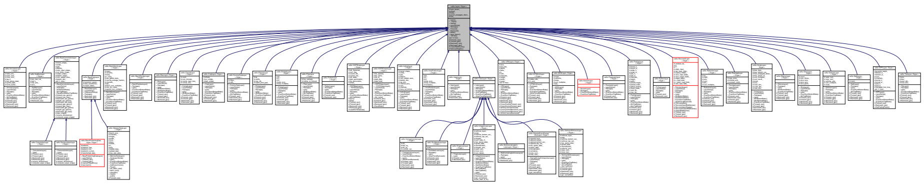
- include/caffe/internal_thread.hpp
- src/caffe/internal_thread.cpp

5.54 caffe::Layer< Dtype > Class Template Reference

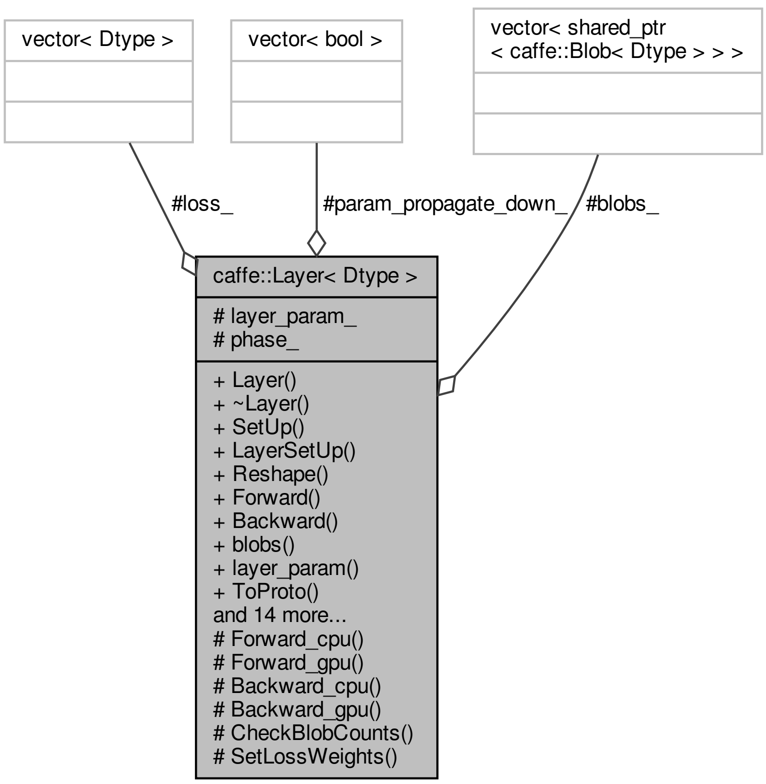
An interface for the units of computation which can be composed into a Net.

`#include <layer.hpp>`

Inheritance diagram for caffe::Layer< Dtype >:



Collaboration diagram for caffe::Layer< Dtype >:



Public Member Functions

- Layer (const LayerParameter &param)
- void SetUp (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Implements common layer setup functionality.
- virtual void LayerSetUp (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void Reshape (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)=0

  Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.
- Dtype Forward (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Given the bottom blobs, compute the top blobs and the loss.
- void Backward (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Given the top blob error gradients, compute the bottom blob error gradients.
- vector< shared_ptr< Blob< Dtype > > > & blobs ()

  Returns the vector of learnable parameter blobs.
- const LayerParameter & layer_param () const

  Returns the layer parameter.
- virtual void ToProto (LayerParameter *param, bool write_diff=false)

  Writes the layer parameter to a protocol buffer.
- Dtype loss (const int top_index) const

  Returns the scalar loss associated with a top blob at a given index.
- void set_loss (const int top_index, const Dtype value)

  Sets the loss associated with a top blob at a given index.
- virtual const char * type () const

  Returns the layer type.
- virtual int ExactNumBottomBlobs () const

  Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.
- virtual int MinBottomBlobs () const

  Returns the minimum number of bottom blobs required by the layer, or -1 if no minimum number is required.
- virtual int MaxBottomBlobs () const

  Returns the maximum number of bottom blobs required by the layer, or -1 if no maximum number is required.
- virtual int ExactNumTopBlobs () const

  Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.
- virtual int MinTopBlobs () const

  Returns the minimum number of top blobs required by the layer, or -1 if no minimum number is required.
- virtual int MaxTopBlobs () const

  Returns the maximum number of top blobs required by the layer, or -1 if no maximum number is required.
- virtual bool EqualNumBottomTopBlobs () const

  Returns true if the layer requires an equal number of bottom and top blobs.
- virtual bool AutoTopBlobs () const

  Return whether "anonymous" top blobs are created automatically by the layer.
- virtual bool AllowForceBackward (const int bottom_index) const

  Return whether to allow force_backward for a given bottom blob index.
- bool param_propagate_down (const int param_id)

  Specifies whether the layer should compute gradients w.r.t. a parameter at a particular index given by param_id.
- void set_param_propagate_down (const int param_id, const bool value)

  Sets whether the layer should compute gradients w.r.t. a parameter at a particular index given by param_id.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)=0

  Using the CPU device, compute the layer output.
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)=0

  Using the CPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.
- virtual void CheckBlobCounts (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)
- void SetLossWeights (const vector< Blob< Dtype > *> &top)

Protected Attributes

- LayerParameter layer_param_
- Phase phase_
- vector< shared_ptr< Blob< Dtype > > > blobs_
- vector< bool > param_propagate_down_
- vector< Dtype > loss_

5.54.1 Detailed Description

template<typename Dtype>
class caffe::Layer< Dtype >

An interface for the units of computation which can be composed into a Net.

Layers must implement a Forward function, in which they take their input (bottom) Blobs (if any) and compute their output Blobs (if any). They may also implement a Backward function, in which they compute the error gradients with respect to their input Blobs, given the error gradients with their output Blobs.

5.54.2 Constructor & Destructor Documentation

5.54.2.1 Layer()

```
template<typename Dtype >
caffe::Layer< Dtype >::Layer (
            const LayerParameter & param )  [inline], [explicit]
```

You should not implement your own constructor. Any set up code should go to SetUp(), where the dimensions of the bottom blobs are provided to the layer.

5.54.3 Member Function Documentation

5.54.3.1 AllowForceBackward()

```
template<typename Dtype >
virtual bool caffe::Layer< Dtype >::AllowForceBackward (
            const int bottom_index ) const  [inline], [virtual]
```

Return whether to allow force_backward for a given bottom blob index.

If AllowForceBackward(i) == false, we will ignore the force_backward setting and backpropagate to blob i only if it needs gradient information (as is done when force_backward == false).

Reimplemented in caffe::LSTMUnitLayer< Dtype >, caffe::RecurrentLayer< Dtype >, caffe::EuclideanLossLayer< Dtype >, caffe::ContrastiveLossLayer< Dtype >, and caffe::LossLayer< Dtype >.

5.54.3.2 AutoTopBlobs()

```
template<typename Dtype >
virtual bool caffe::Layer< Dtype >::AutoTopBlobs ( ) const  [inline], [virtual]
```

Return whether "anonymous" top blobs are created automatically by the layer.

If this method returns true, Net::Init will create enough "anonymous" top blobs to fulfill the requirement specified by ExactNumTopBlobs() or MinTopBlobs().

Reimplemented in caffe::LossLayer< Dtype >.

5.54.3.3 Backward()

```
template<typename Dtype >
void caffe::Layer< Dtype >::Backward (
            const vector< Blob< Dtype > *> & top,
            const vector< bool > & propagate_down,
            const vector< Blob< Dtype > *> & bottom )  [inline]
```

Given the top blob error gradients, compute the bottom blob error gradients.

Parameters

top	the output blobs, whose diff fields store the gradient of the error with respect to themselves
propagate_down	a vector with equal length to bottom, with each index indicating whether to propagate the error gradients down to the bottom blob at the corresponding index
bottom	the input blobs, whose diff fields will store the gradient of the error with respect to themselves after Backward is run

The Backward wrapper calls the relevant device wrapper function (Backward_cpu or Backward_gpu) to compute the bottom blob diffs given the top blob diffs.

Your layer should implement Backward_cpu and (optionally) Backward_gpu.

5.54.3.4 CheckBlobCounts()

```
template<typename Dtype >
virtual void caffe::Layer< Dtype >::CheckBlobCounts (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [inline], [protected], [virtual]
```

Called by the parent Layer's SetUp to check that the number of bottom and top Blobs provided as input match the expected numbers specified by the {ExactNum,Min,Max}{Bottom,Top}Blobs() functions.

5.54.3.5 EqualNumBottomTopBlobs()

```
template<typename Dtype >
virtual bool caffe::Layer< Dtype >::EqualNumBottomTopBlobs ( ) const  [inline], [virtual]
```

Returns true if the layer requires an equal number of bottom and top blobs.

This method should be overridden to return true if your layer expects an equal number of bottom and top blobs.

Reimplemented in caffe::BaseConvolutionLayer< Dtype >.

5.54.3.6 ExactNumBottomBlobs()

```
template<typename Dtype >
virtual int caffe::Layer< Dtype >::ExactNumBottomBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of bottom blobs.

Reimplemented in caffe::LSTMUnitLayer< Dtype >, caffe::InfogainLossLayer< Dtype >, caffe::BatchNormLayer< Dtype >, caffe::ArgMaxLayer< Dtype >, caffe::ContrastiveLossLayer< Dtype >, caffe::AccuracyLayer< Dtype >, caffe::HDF5OutputLayer< Dtype >, caffe::HDF5DataLayer< Dtype >, caffe::WindowDataLayer< Dtype >, caffe::AbsValLayer< Dtype >, caffe::LRNLayer< Dtype >, caffe::ImageDataLayer< Dtype >, caffe::LossLayer< Dtype >, caffe::CropLayer< Dtype >, caffe::FlattenLayer< Dtype >, caffe::DummyDataLayer< Dtype >, caffe::↩ EmbedLayer< Dtype >, caffe::Im2colLayer< Dtype >, caffe::InputLayer< Dtype >, caffe::ReductionLayer< Dtype >, caffe::BatchReindexLayer< Dtype >, caffe::InnerProductLayer< Dtype >, caffe::ParameterLayer< Dtype >, caffe::ReshapeLayer< Dtype >, caffe::SliceLayer< Dtype >, caffe::SPPLayer< Dtype >, caffe::MemoryData↩ Layer< Dtype >, caffe::PoolingLayer< Dtype >, caffe::SplitLayer< Dtype >, caffe::MVNLayer< Dtype >, caffe::↩ NeuronLayer< Dtype >, caffe::SoftmaxLayer< Dtype >, caffe::DataLayer< Dtype >, and caffe::TileLayer< Dtype >.

5.54.3.7 ExactNumTopBlobs()

```
template<typename Dtype >
virtual int caffe::Layer< Dtype >::ExactNumTopBlobs ( ) const [inline], [virtual]
```

Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of top blobs.

Reimplemented in caffe::LSTMUnitLayer< Dtype >, caffe::InfogainLossLayer< Dtype >, caffe::SoftmaxWithLossLayer< Dtype >, caffe::BatchNormLayer< Dtype >, caffe::ArgMaxLayer< Dtype >, caffe::RecurrentLayer< Dtype >, caffe::LossLayer< Dtype >, caffe::ScaleLayer< Dtype >, caffe::HDF5OutputLayer< Dtype >, caffe::WindowDataLayer< Dtype >, caffe::AbsValLayer< Dtype >, caffe::BiasLayer< Dtype >, caffe::LRNLayer< Dtype >, caffe::ImageDataLayer< Dtype >, caffe::CropLayer< Dtype >, caffe::FlattenLayer< Dtype >, caffe::EmbedLayer< Dtype >, caffe::Im2colLayer< Dtype >, caffe::ReductionLayer< Dtype >, caffe::BatchReindexLayer< Dtype >, caffe::EltwiseLayer< Dtype >, caffe::InnerProductLayer< Dtype >, caffe::ParameterLayer< Dtype >, caffe::ReshapeLayer< Dtype >, caffe::SPPLayer< Dtype >, caffe::MemoryDataLayer< Dtype >, caffe::ConcatLayer< Dtype >, caffe::MVNLayer< Dtype >, caffe::NeuronLayer< Dtype >, caffe::SoftmaxLayer< Dtype >, caffe::SilenceLayer< Dtype >, and caffe::TileLayer< Dtype >.

5.54.3.8 Forward()

```
template<typename Dtype >
Dtype caffe::Layer< Dtype >::Forward (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [inline]
```

Given the bottom blobs, compute the top blobs and the loss.

Parameters

bottom	the input blobs, whose data fields store the input data for this layer
top	the preshaped output blobs, whose data fields will store this layers' outputs

Returns

The total loss from the layer.

The Forward wrapper calls the relevant device wrapper function (Forward_cpu or Forward_gpu) to compute the top blob values given the bottom blobs. If the layer has any non-zero loss_weights, the wrapper then computes and returns the loss.

Your layer should implement Forward_cpu and (optionally) Forward_gpu.

5.54.3.9 LayerSetUp()

```
template<typename Dtype >
virtual void caffe::Layer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [inline], [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented in caffe::BasePrefetchingDataLayer< Dtype >, caffe::SoftmaxWithLossLayer< Dtype >, caffe::InfogainLossLayer< Dtype >, caffe::SigmoidCrossEntropyLossLayer< Dtype >, caffe::BatchNormLayer< Dtype >, caffe::ContrastiveLossLayer< Dtype >, caffe::ArgMaxLayer< Dtype >, caffe::DropoutLayer< Dtype >, caffe::PReLULayer< Dtype >, caffe::SwishLayer< Dtype >, caffe::ExpLayer< Dtype >, caffe::LogLayer< Dtype >, caffe::AccuracyLayer< Dtype >, caffe::PowerLayer< Dtype >, caffe::RecurrentLayer< Dtype >, caffe::ScaleLayer< Dtype >, caffe::AbsValLayer< Dtype >, caffe::HDF5OutputLayer< Dtype >, caffe::ThresholdLayer< Dtype >, caffe::HDF5DataLayer< Dtype >, caffe::BaseDataLayer< Dtype >, caffe::LossLayer< Dtype >, caffe::LRNLayer< Dtype >, caffe::BiasLayer< Dtype >, caffe::CropLayer< Dtype >, caffe::EmbedLayer< Dtype >, caffe::Im2colLayer< Dtype >, caffe::ReductionLayer< Dtype >, caffe::DummyDataLayer< Dtype >, caffe::EltwiseLayer< Dtype >, caffe::FilterLayer< Dtype >, caffe::InnerProductLayer< Dtype >, caffe::InputLayer< Dtype >, caffe::ReshapeLayer< Dtype >, caffe::SliceLayer< Dtype >, caffe::SPPLayer< Dtype >, caffe::BaseConvolutionLayer< Dtype >, caffe::PoolingLayer< Dtype >, caffe::ConcatLayer< Dtype >, caffe::PythonLayer< Dtype >, and caffe::ParameterLayer< Dtype >.

5.54.3.10 MaxBottomBlobs()

```
template<typename Dtype >
virtual int caffe::Layer< Dtype >::MaxBottomBlobs ( ) const [inline], [virtual]
```

Returns the maximum number of bottom blobs required by the layer, or -1 if no maximum number is required.

This method should be overridden to return a non-negative value if your layer expects some maximum number of bottom blobs.

Reimplemented in caffe::InfogainLossLayer< Dtype >, caffe::RecurrentLayer< Dtype >, caffe::ScaleLayer< Dtype >, and caffe::BiasLayer< Dtype >.

5.54.3.11 MaxTopBlobs()

```
template<typename Dtype >
virtual int caffe::Layer< Dtype >::MaxTopBlobs ( ) const [inline], [virtual]
```

Returns the maximum number of top blobs required by the layer, or -1 if no maximum number is required.

This method should be overridden to return a non-negative value if your layer expects some maximum number of top blobs.

Reimplemented in caffe::InfogainLossLayer< Dtype >, caffe::SoftmaxWithLossLayer< Dtype >, caffe::AccuracyLayer< Dtype >, caffe::PoolingLayer< Dtype >, and caffe::DataLayer< Dtype >.

5.54.3.12 MinBottomBlobs()

```
template<typename Dtype >
virtual int caffe::Layer< Dtype >::MinBottomBlobs ( ) const [inline], [virtual]
```

Returns the minimum number of bottom blobs required by the layer, or -1 if no minimum number is required.

This method should be overridden to return a non-negative value if your layer expects some minimum number of bottom blobs.

Reimplemented in caffe::InfogainLossLayer< Dtype >, caffe::RecurrentLayer< Dtype >, caffe::ScaleLayer< Dtype >, caffe::BiasLayer< Dtype >, caffe::EltwiseLayer< Dtype >, caffe::FilterLayer< Dtype >, caffe::BaseConvolutionLayer< Dtype >, caffe::ConcatLayer< Dtype >, and caffe::SilenceLayer< Dtype >.

5.54.3.13 MinTopBlobs()

```
template<typename Dtype >
virtual int caffe::Layer< Dtype >::MinTopBlobs ( ) const [inline], [virtual]
```

Returns the minimum number of top blobs required by the layer, or -1 if no minimum number is required.

This method should be overridden to return a non-negative value if your layer expects some minimum number of top blobs.

Reimplemented in caffe::InfogainLossLayer< Dtype >, caffe::SoftmaxWithLossLayer< Dtype >, caffe::AccuracyLayer< Dtype >, caffe::HDF5DataLayer< Dtype >, caffe::DummyDataLayer< Dtype >, caffe::InputLayer< Dtype >, caffe::FilterLayer< Dtype >, caffe::SliceLayer< Dtype >, caffe::PoolingLayer< Dtype >, caffe::BaseConvolutionLayer< Dtype >, caffe::SplitLayer< Dtype >, and caffe::DataLayer< Dtype >.

5.54.3.14 param_propagate_down()

```
template<typename Dtype >
bool caffe::Layer< Dtype >::param_propagate_down (
            const int param_id ) [inline]
```

Specifies whether the layer should compute gradients w.r.t. a parameter at a particular index given by param_id.

You can safely ignore false values and always compute gradients for all parameters, but possibly with wasteful computation.

5.54.3.15 Reshape()

```
template<typename Dtype >
virtual void caffe::Layer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [pure virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Implemented in caffe::LSTMUnitLayer< Dtype >, caffe::SoftmaxWithLossLayer< Dtype >, caffe::InfogainLossLayer< Dtype >, caffe::SigmoidCrossEntropyLossLayer< Dtype >, caffe::MultinomialLogisticLossLayer< Dtype >, caffe::BatchNormLayer< Dtype >, caffe::EuclideanLossLayer< Dtype >, caffe::ArgMaxLayer< Dtype >, caffe::PReLULayer< Dtype >, caffe::DropoutLayer< Dtype >, caffe::SwishLayer< Dtype >, caffe::AccuracyLayer< Dtype >, caffe::BaseDataLayer< Dtype >, caffe::HDF5OutputLayer< Dtype >, caffe::PythonLayer< Dtype >, caffe::RecurrentLayer< Dtype >, caffe::ScaleLayer< Dtype >, caffe::HDF5DataLayer< Dtype >, caffe::LossLayer< Dtype >, caffe::LRNLayer< Dtype >, caffe::BiasLayer< Dtype >, caffe::CropLayer< Dtype >, caffe::FlattenLayer< Dtype >, caffe::DummyDataLayer< Dtype >, caffe::EmbedLayer< Dtype >, caffe::Im2colLayer< Dtype >, caffe::InputLayer< Dtype >, caffe::ParameterLayer< Dtype >, caffe::ReductionLayer< Dtype >, caffe::BatchReindexLayer< Dtype >, caffe::EltwiseLayer< Dtype >, caffe::FilterLayer< Dtype >, caffe::InnerProductLayer< Dtype >, caffe::ReshapeLayer< Dtype >, caffe::SliceLayer< Dtype >, caffe::SPPLayer< Dtype >, caffe::BaseConvolutionLayer< Dtype >, caffe::PoolingLayer< Dtype >, caffe::ConcatLayer< Dtype >, caffe::NeuronLayer< Dtype >, caffe::SplitLayer< Dtype >, caffe::MVNLayer< Dtype >, caffe::SoftmaxLayer< Dtype >, caffe::SilenceLayer< Dtype >, and caffe::TileLayer< Dtype >.

5.54.3.16 SetLossWeights()

```
template<typename Dtype >
void caffe::Layer< Dtype >::SetLossWeights (
            const vector< Blob< Dtype > *> & top )  [inline], [protected]
```

Called by SetUp to initialize the weights associated with any top blobs in the loss function. Store non-zero loss weights in the diff blob.

5.54.3.17 SetUp()

```
template<typename Dtype >
void caffe::Layer< Dtype >::SetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [inline]
```

Implements common layer setup functionality.

Parameters

bottom	the preshaped input blobs
top	the allocated but unshaped output blobs, to be shaped by Reshape

Checks that the number of bottom and top blobs is correct. Calls LayerSetUp to do special layer setup for individual layer types, followed by Reshape to set up sizes of top blobs and internal buffers. Sets up the loss weight multiplier blobs for any non-zero loss weights. This method may not be overridden.

5.54.4 Member Data Documentation

5.54.4.1 blobs_

```
template<typename Dtype >
vector<shared_ptr<Blob<Dtype> > > caffe::Layer< Dtype >::blobs_ [protected]
```

The vector that stores the learnable parameters as a set of blobs.

5.54.4.2 layer_param_

```
template<typename Dtype >
LayerParameter caffe::Layer< Dtype >::layer_param_ [protected]
```

The protobuf that stores the layer parameters

5.54.4.3 loss_

```
template<typename Dtype >
vector<Dtype> caffe::Layer< Dtype >::loss_ [protected]
```

The vector that indicates whether each top blob has a non-zero weight in the objective function.

5.54.4.4 param_propagate_down_

```
template<typename Dtype >
vector<bool> caffe::Layer< Dtype >::param_propagate_down_ [protected]
```

Vector indicating whether to compute the diff of each param blob.

5.54.4.5 phase_

```
template<typename Dtype >
Phase caffe::Layer< Dtype >::phase_ [protected]
```

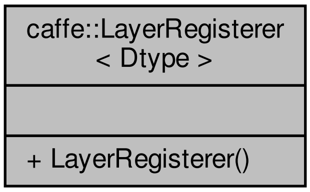
The phase: TRAIN or TEST

The documentation for this class was generated from the following file:

- include/caffe/layer.hpp

5.55 caffe::LayerRegisterer< Dtype > Class Template Reference

Collaboration diagram for caffe::LayerRegisterer< Dtype >:



Public Member Functions

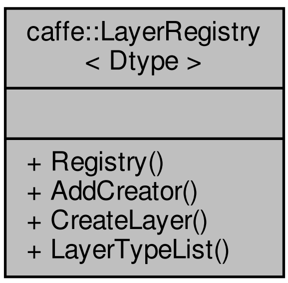
- **LayerRegisterer** (const string &type, shared_ptr< Layer< Dtype > >(∗creator)(const LayerParameter &))

The documentation for this class was generated from the following file:

- include/caffe/layer_factory.hpp

5.56 caffe::LayerRegistry< Dtype > Class Template Reference

Collaboration diagram for caffe::LayerRegistry< Dtype >:



Public Types

- typedef shared_ptr< Layer< Dtype > >(∗ **Creator**) (const LayerParameter &)
- typedef std::map< string, Creator > **CreatorRegistry**

Static Public Member Functions

- static CreatorRegistry & **Registry** ()
- static void **AddCreator** (const string &type, Creator creator)
- static shared_ptr$<$ Layer$<$ Dtype $>$ $>$ **CreateLayer** (const LayerParameter &param)
- static vector$<$ string $>$ **LayerTypeList** ()

The documentation for this class was generated from the following file:

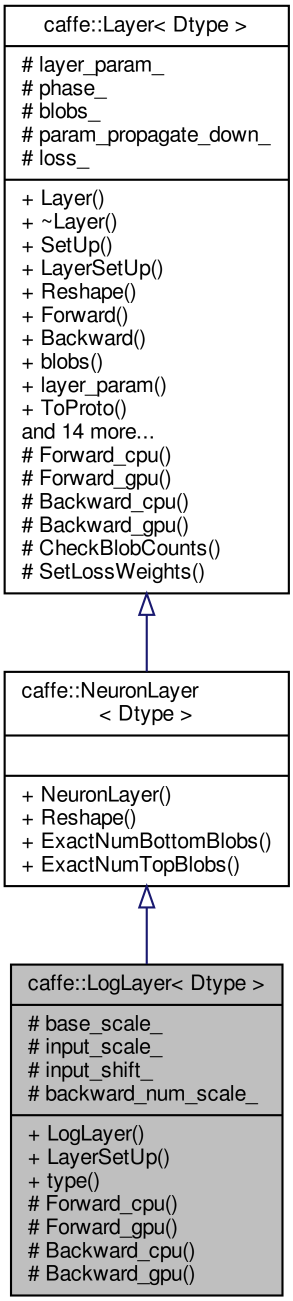
- include/caffe/layer_factory.hpp

5.57 caffe::LogLayer$<$ Dtype $>$ Class Template Reference

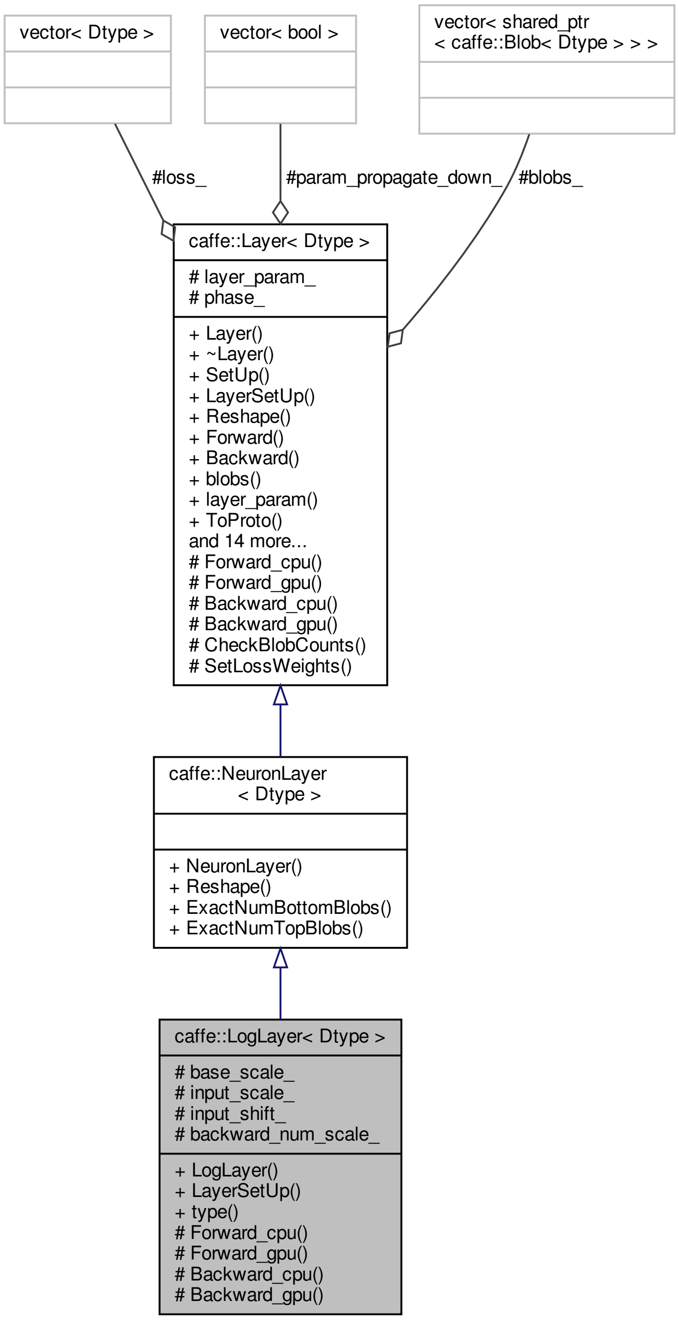
Computes $y = log_\gamma(\alpha x + \beta)$, as specified by the scale α, shift β, and base γ.

```
#include <log_layer.hpp>
```

Inheritance diagram for caffe::LogLayer< Dtype >:



Collaboration diagram for caffe::LogLayer< Dtype >:



Public Member Functions

- LogLayer (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

    Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual const char ∗ type () const

    Returns the layer type.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

    Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

    Computes the error gradient w.r.t. the exp inputs.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

    Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Protected Attributes

- Dtype **base_scale_**
- Dtype **input_scale_**
- Dtype **input_shift_**
- Dtype **backward_num_scale_**

5.57.1 Detailed Description

template<typename Dtype>
class caffe::LogLayer< Dtype >

Computes $y = log_{\gamma}(\alpha x + \beta)$, as specified by the scale α, shift β, and base γ.

5.57.2 Constructor & Destructor Documentation

5.57.2.1 LogLayer()

```
template<typename Dtype >
caffe::LogLayer< Dtype >::LogLayer (
            const LayerParameter & param )  [inline], [explicit]
```

Parameters

param	provides LogParameter log_param, with LogLayer options: • scale (**optional**, default 1) the scale α • shift (**optional**, default 0) the shift β • base (**optional**, default -1 for a value of $e \approx 2.718$) the base γ

5.57.3 Member Function Documentation

5.57.3.1 Backward_cpu()

```
template<typename Dtype >
void caffe::LogLayer< Dtype >::Backward_cpu (
            const vector< Blob< Dtype > *> & top,
            const vector< bool > & propagate_down,
            const vector< Blob< Dtype > *> & bottom )  [protected], [virtual]
```

Computes the error gradient w.r.t. the exp inputs.

Parameters

top	output Blob vector (length 1), providing the error gradient with respect to the outputs 1. $(N \times C \times H \times W)$ containing error gradients $\frac{\partial E}{\partial y}$ with respect to computed outputs y
propagate_down	see Layer::Backward.
bottom	input Blob vector (length 1) 1. $(N \times C \times H \times W)$ the inputs x; Backward fills their diff with gradients $\frac{\partial E}{\partial x} = \frac{\partial E}{\partial y} y \alpha \log_e(gamma)$ if propagate_down[0]

Implements caffe::Layer< Dtype >.

5.57.3.2 Forward_cpu()

```
template<typename Dtype >
void caffe::LogLayer< Dtype >::Forward_cpu (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [protected], [virtual]
```

Parameters

bottom	input Blob vector (length 1) 1. $(N \times C \times H \times W)$ the inputs x
top	output Blob vector (length 1) 1. $(N \times C \times H \times W)$ the computed outputs $y = log_\gamma(\alpha x + \beta)$

Implements caffe::Layer< Dtype >.

5.57.3.3 LayerSetUp()

```
template<typename Dtype >
void caffe::LogLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

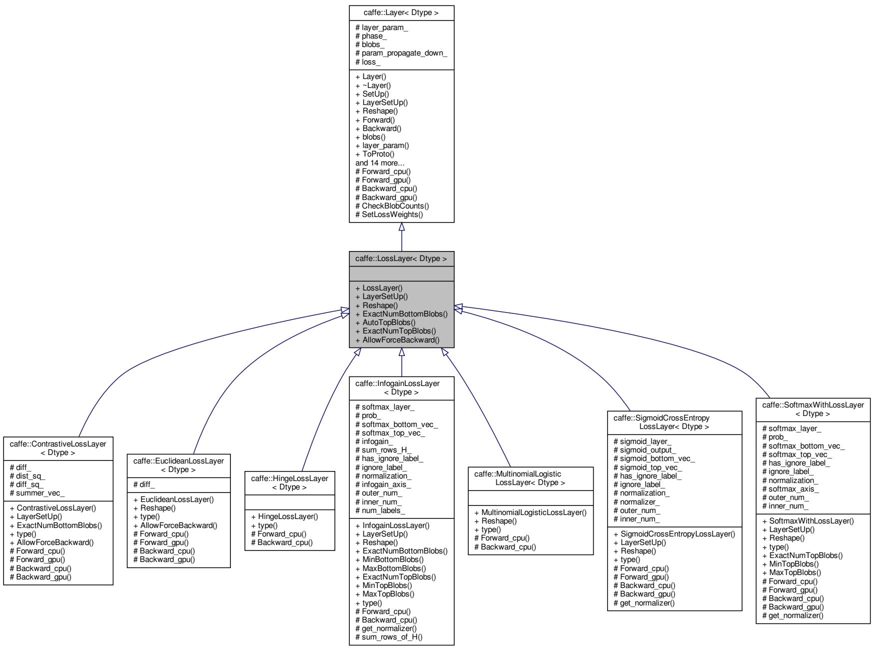
- include/caffe/layers/log_layer.hpp
- src/caffe/layers/log_layer.cpp

5.58 caffe::LossLayer< Dtype > Class Template Reference

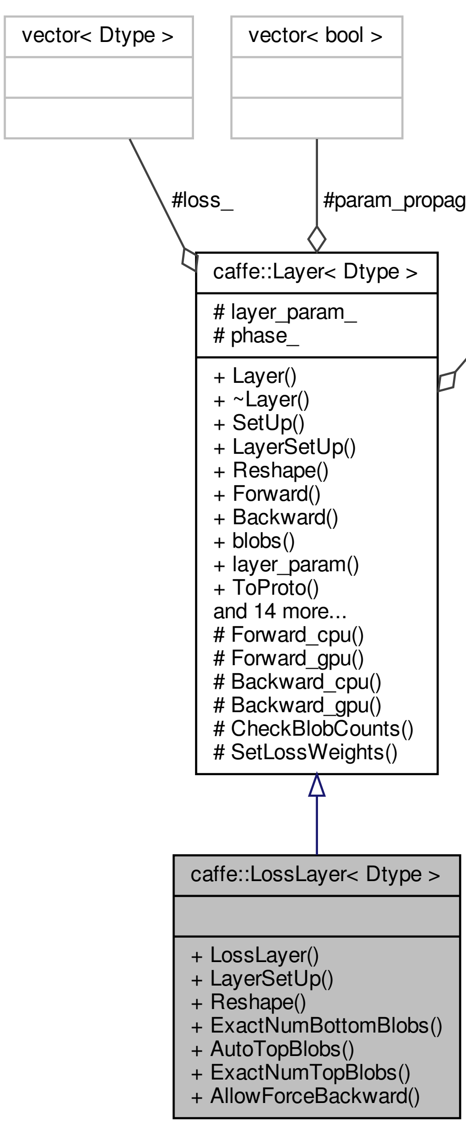
An interface for Layers that take two Blobs as input – usually (1) predictions and (2) ground-truth labels – and output a singleton Blob representing the loss.

`#include <loss_layer.hpp>`

Inheritance diagram for caffe::LossLayer< Dtype >:



Collaboration diagram for caffe::LossLayer< Dtype >:



Public Member Functions

- **LossLayer** (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void Reshape (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.
- virtual int ExactNumBottomBlobs () const

  Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.
- virtual bool AutoTopBlobs () const

  For convenience and backwards compatibility, instruct the Net to automatically allocate a single top Blob for LossLayers, into which they output their singleton loss, (even if the user didn't specify one in the prototxt, etc.).

- virtual int ExactNumTopBlobs () const

  Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.
- virtual bool AllowForceBackward (const int bottom_index) const

Additional Inherited Members

5.58.1 Detailed Description

template<typename Dtype>
class caffe::LossLayer< Dtype >

An interface for Layers that take two Blobs as input – usually (1) predictions and (2) ground-truth labels – and output a singleton Blob representing the loss.

LossLayers are typically only capable of backpropagating to their first input – the predictions.

5.58.2 Member Function Documentation

5.58.2.1 AllowForceBackward()

```
template<typename Dtype >
virtual bool caffe::LossLayer< Dtype >::AllowForceBackward (
            const int bottom_index ) const  [inline], [virtual]
```

We usually cannot backpropagate to the labels; ignore force_backward for these inputs.

Reimplemented from caffe::Layer< Dtype >.

Reimplemented in caffe::EuclideanLossLayer< Dtype >, and caffe::ContrastiveLossLayer< Dtype >.

5.58.2.2 ExactNumBottomBlobs()

```
template<typename Dtype >
virtual int caffe::LossLayer< Dtype >::ExactNumBottomBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

Reimplemented in caffe::InfogainLossLayer< Dtype >, and caffe::ContrastiveLossLayer< Dtype >.

5.58.2.3 ExactNumTopBlobs()

```
template<typename Dtype >
virtual int caffe::LossLayer< Dtype >::ExactNumTopBlobs ( ) const [inline], [virtual]
```

Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

Reimplemented in caffe::InfogainLossLayer< Dtype >, and caffe::SoftmaxWithLossLayer< Dtype >.

5.58.2.4 LayerSetUp()

```
template<typename Dtype >
void caffe::LossLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

Reimplemented in caffe::SoftmaxWithLossLayer< Dtype >, caffe::InfogainLossLayer< Dtype >, caffe::SigmoidCrossEntropyLossLayer< Dtype >, and caffe::ContrastiveLossLayer< Dtype >.

5.58.2.5 Reshape()

```
template<typename Dtype >
void caffe::LossLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Implements caffe::Layer< Dtype >.

Reimplemented in caffe::SoftmaxWithLossLayer< Dtype >, caffe::InfogainLossLayer< Dtype >, caffe::Sigmoid↩ CrossEntropyLossLayer< Dtype >, caffe::MultinomialLogisticLossLayer< Dtype >, and caffe::EuclideanLoss↩ Layer< Dtype >.

The documentation for this class was generated from the following files:

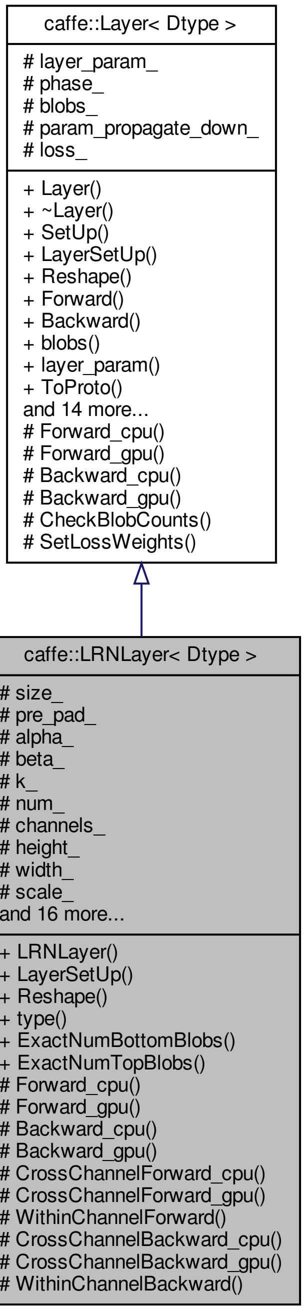
- include/caffe/layers/loss_layer.hpp
- src/caffe/layers/loss_layer.cpp

5.59 caffe::LRNLayer< Dtype > Class Template Reference

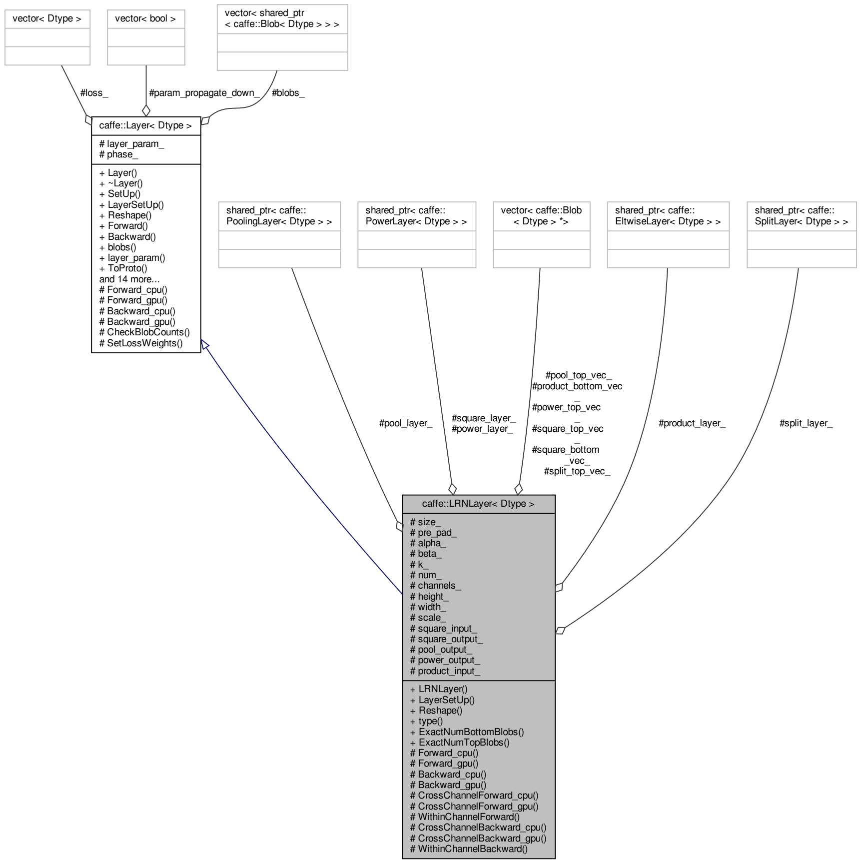
Normalize the input in a local region across or within feature maps.

```
#include <lrn_layer.hpp>
```

Inheritance diagram for caffe::LRNLayer< Dtype >:



Collaboration diagram for caffe::LRNLayer< Dtype >:



Public Member Functions

- **LRNLayer** (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void Reshape (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.
- virtual const char * type () const

  Returns the layer type.
- virtual int ExactNumBottomBlobs () const

  Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.
- virtual int ExactNumTopBlobs () const

  Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the CPU device, compute the layer output.
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.

- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the CPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.
- virtual void **CrossChannelForward_cpu** (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)
- virtual void **CrossChannelForward_gpu** (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)
- virtual void **WithinChannelForward** (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)
- virtual void **CrossChannelBackward_cpu** (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)
- virtual void **CrossChannelBackward_gpu** (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)
- virtual void **WithinChannelBackward** (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

Protected Attributes

- int **size_**
- int **pre_pad_**
- Dtype **alpha_**
- Dtype **beta_**
- Dtype **k_**
- int **num_**
- int **channels_**
- int **height_**
- int **width_**
- Blob< Dtype > **scale_**
- shared_ptr< SplitLayer< Dtype > > **split_layer_**
- vector< Blob< Dtype > * > **split_top_vec_**
- shared_ptr< PowerLayer< Dtype > > **square_layer_**
- Blob< Dtype > **square_input_**
- Blob< Dtype > **square_output_**
- vector< Blob< Dtype > * > **square_bottom_vec_**
- vector< Blob< Dtype > * > **square_top_vec_**
- shared_ptr< PoolingLayer< Dtype > > **pool_layer_**
- Blob< Dtype > **pool_output_**
- vector< Blob< Dtype > * > **pool_top_vec_**
- shared_ptr< PowerLayer< Dtype > > **power_layer_**
- Blob< Dtype > **power_output_**
- vector< Blob< Dtype > * > **power_top_vec_**
- shared_ptr< EltwiseLayer< Dtype > > **product_layer_**
- Blob< Dtype > **product_input_**
- vector< Blob< Dtype > * > **product_bottom_vec_**

5.59.1 Detailed Description

template<typename Dtype>
class caffe::LRNLayer< Dtype >

Normalize the input in a local region across or within feature maps.

TODO(dox): thorough documentation for Forward, Backward, and proto params.

5.59.2 Member Function Documentation

5.59.2.1 ExactNumBottomBlobs()

```
template<typename Dtype >
virtual int caffe::LRNLayer< Dtype >::ExactNumBottomBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.59.2.2 ExactNumTopBlobs()

```
template<typename Dtype >
virtual int caffe::LRNLayer< Dtype >::ExactNumTopBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.59.2.3 LayerSetUp()

```
template<typename Dtype >
void caffe::LRNLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

5.59.2.4 Reshape()

```
template<typename Dtype >
void caffe::LRNLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

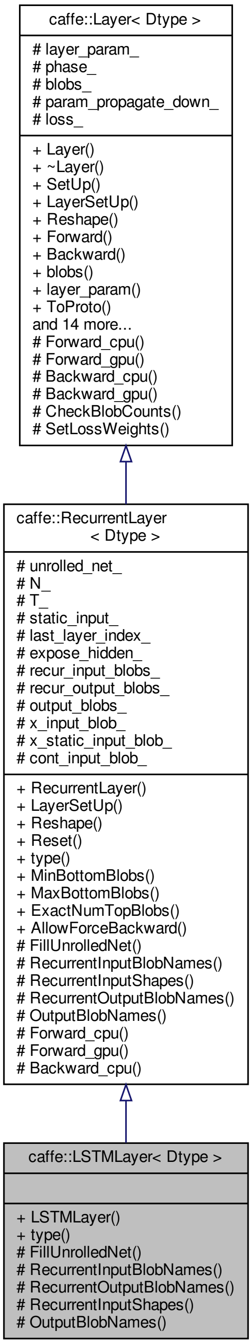
- include/caffe/layers/lrn_layer.hpp
- src/caffe/layers/lrn_layer.cpp

5.60 caffe::LSTMLayer< Dtype > Class Template Reference

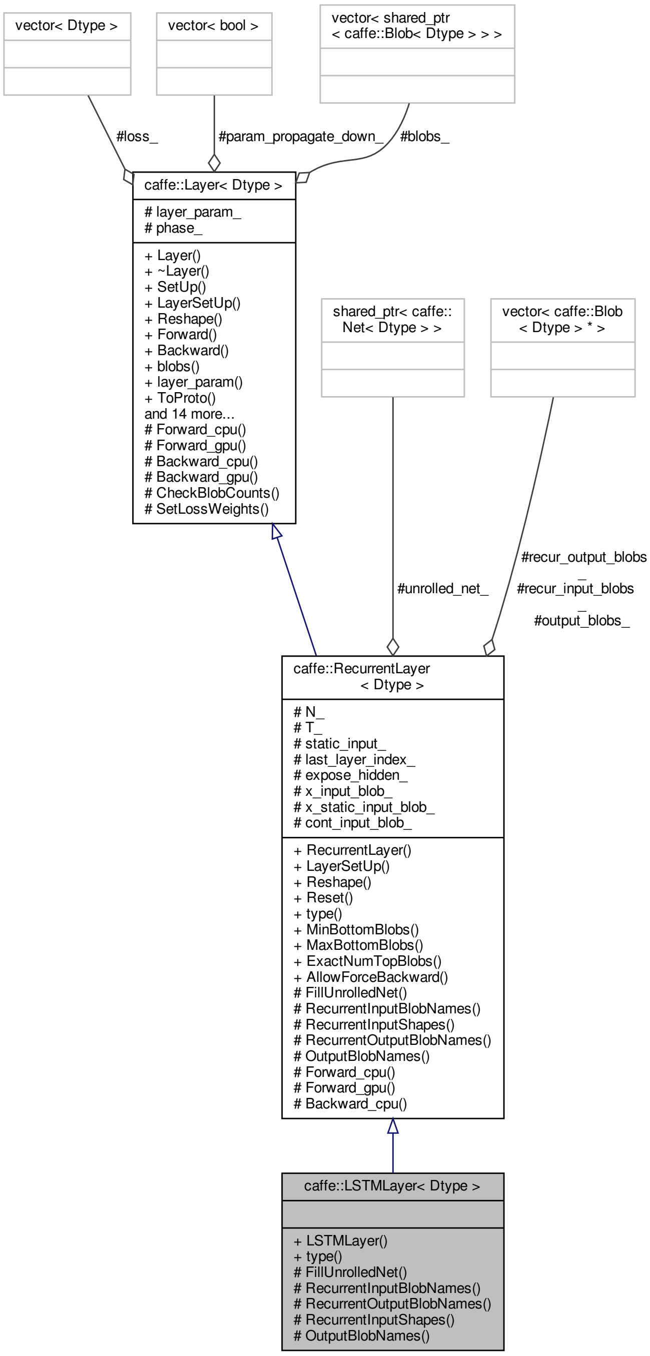
Processes sequential inputs using a "Long Short-Term Memory" (LSTM) [1] style recurrent neural network (RNN). Implemented by unrolling the LSTM computation through time.

`#include <lstm_layer.hpp>`

Inheritance diagram for caffe::LSTMLayer< Dtype >:



Collaboration diagram for caffe::LSTMLayer< Dtype >:



Public Member Functions

- **LSTMLayer** (const LayerParameter &param)
- virtual const char $*$ type () const

  Returns the layer type.

Protected Member Functions

- virtual void FillUnrolledNet (NetParameter *net_param) const

  Fills net_param with the recurrent network architecture. Subclasses should define this – see RNNLayer and LSTM↵ Layer for examples.
- virtual void RecurrentInputBlobNames (vector< string > *names) const

  Fills names with the names of the 0th timestep recurrent input Blob&s. Subclasses should define this – see RNNLayer and LSTMLayer for examples.
- virtual void RecurrentOutputBlobNames (vector< string > *names) const

  Fills names with the names of the Tth timestep recurrent output Blob&s. Subclasses should define this – see RNN↵ Layer and LSTMLayer for examples.
- virtual void RecurrentInputShapes (vector< BlobShape > *shapes) const

  Fills shapes with the shapes of the recurrent input Blob&s. Subclasses should define this – see RNNLayer and LSTMLayer for examples.
- virtual void OutputBlobNames (vector< string > *names) const

  Fills names with the names of the output blobs, concatenated across all timesteps. Should return a name for each top Blob. Subclasses should define this – see RNNLayer and LSTMLayer for examples.

Additional Inherited Members

5.60.1 Detailed Description

template<typename Dtype>
class caffe::LSTMLayer< Dtype >

Processes sequential inputs using a "Long Short-Term Memory" (LSTM) [1] style recurrent neural network (RNN). Implemented by unrolling the LSTM computation through time.

The specific architecture used in this implementation is as described in "Learning to Execute" [2], reproduced below: i_t := [W_{hi} * h_{t-1} + W_{xi} * x_t + b_i] f_t := [W_{hf} * h_{t-1} + W_{xf} * x_t + b_f] o_t := [W_{ho} * h_{t-1} + W_{xo} * x_t + b_o] g_t := [W_{hg} * h_{t-1} + W_{xg} * x_t + b_g] c_t := (f_t .* c_{t-1}) + (i_t .* g_t) h_t := o_t .* [c_t] In the implementation, the i, f, o, and g computations are performed as a single inner product.

Notably, this implementation lacks the "diagonal" gates, as used in the LSTM architectures described by Alex Graves [3] and others.

[1] Hochreiter, Sepp, and Schmidhuber, Jürgen. "Long short-term memory." Neural Computation 9, no. 8 (1997): 1735-1780.

[2] Zaremba, Wojciech, and Sutskever, Ilya. "Learning to execute." arXiv preprint arXiv:1410.4615 (2014).

[3] Graves, Alex. "Generating sequences with recurrent neural networks." arXiv preprint arXiv:1308.0850 (2013).

The documentation for this class was generated from the following files:

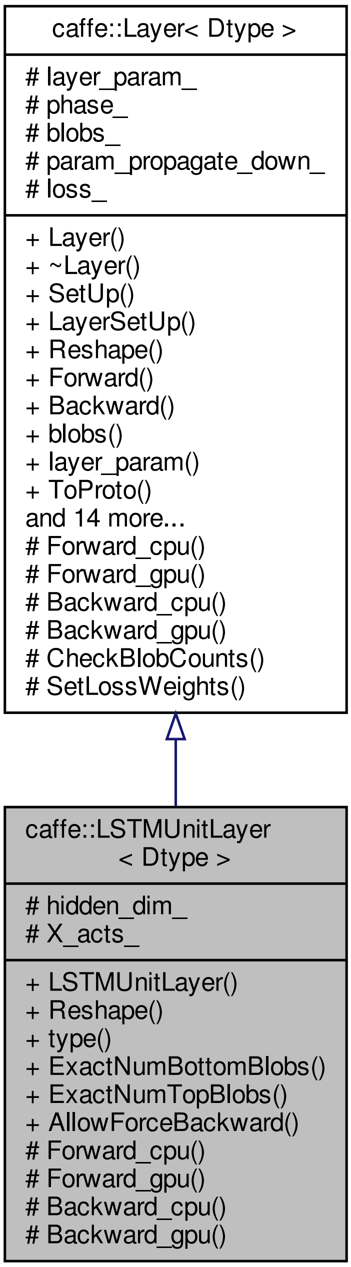
- include/caffe/layers/lstm_layer.hpp
- src/caffe/layers/lstm_layer.cpp

5.61 caffe::LSTMUnitLayer< Dtype > Class Template Reference

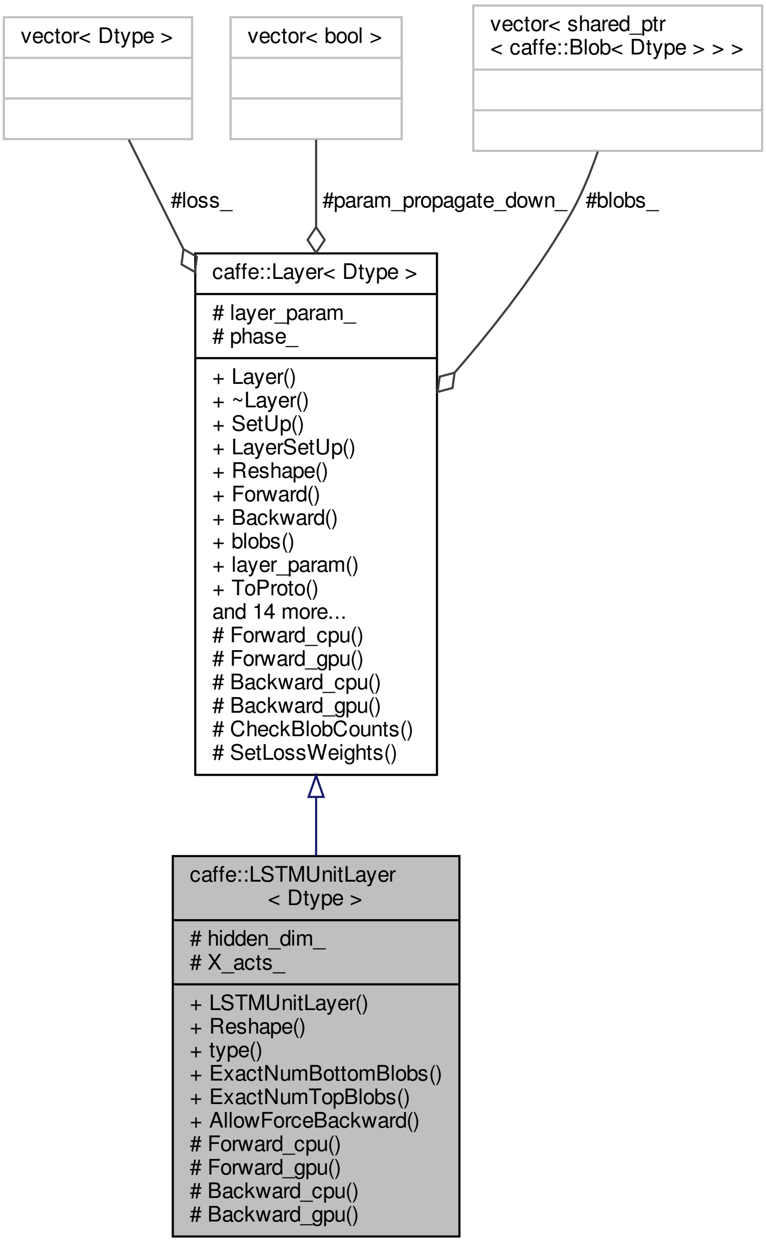
A helper for LSTMLayer: computes a single timestep of the non-linearity of the LSTM, producing the updated cell and hidden states.

```
#include <lstm_layer.hpp>
```

Inheritance diagram for caffe::LSTMUnitLayer< Dtype >:



Collaboration diagram for caffe::LSTMUnitLayer< Dtype >:



Public Member Functions

- **LSTMUnitLayer** (const LayerParameter &param)
- virtual void Reshape (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

    Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.
- virtual const char * type () const

    Returns the layer type.
- virtual int ExactNumBottomBlobs () const

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.
- virtual int ExactNumTopBlobs () const

  Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.
- virtual bool AllowForceBackward (const int bottom_index) const

  Return whether to allow force_backward for a given bottom blob index.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Computes the error gradient w.r.t. the LSTMUnit inputs.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Protected Attributes

- int hidden_dim_

  The hidden and output dimension.
- Blob< Dtype > **X_acts_**

5.61.1 Detailed Description

template<typename Dtype>
class caffe::LSTMUnitLayer< Dtype >

A helper for LSTMLayer: computes a single timestep of the non-linearity of the LSTM, producing the updated cell and hidden states.

5.61.2 Member Function Documentation

5.61.2.1 AllowForceBackward()

```
template<typename Dtype >
virtual bool caffe::LSTMUnitLayer< Dtype >::AllowForceBackward (
            const int bottom_index ) const  [inline], [virtual]
```

Return whether to allow force_backward for a given bottom blob index.

If AllowForceBackward(i) == false, we will ignore the force_backward setting and backpropagate to blob i only if it needs gradient information (as is done when force_backward == false).

Reimplemented from caffe::Layer< Dtype >.

5.61.2.2 Backward_cpu()

```
template<typename Dtype >
void caffe::LSTMUnitLayer< Dtype >::Backward_cpu (
            const vector< Blob< Dtype > *> & top,
            const vector< bool > & propagate_down,
            const vector< Blob< Dtype > *> & bottom ) [protected], [virtual]
```

Computes the error gradient w.r.t. the LSTMUnit inputs.

Parameters

top	output Blob vector (length 2), providing the error gradient with respect to the outputs 1. $(1 \times N \times D)$: containing error gradients $\frac{\partial E}{\partial c_t}$ with respect to the updated cell state c_t 2. $(1 \times N \times D)$: containing error gradients $\frac{\partial E}{\partial h_t}$ with respect to the updated cell state h_t
propagate_down	see Layer::Backward.
bottom	input Blob vector (length 3), into which the error gradients with respect to the LSTMUnit inputs c_{t-1} and the gate inputs are computed. Computatation of the error gradients w.r.t. the sequence indicators is not implemented. 1. $(1 \times N \times D)$ the error gradient w.r.t. the previous timestep cell state c_{t-1} 2. $(1 \times N \times 4D)$ the error gradient w.r.t. the "gate inputs" $[\frac{\partial E}{\partial i_t} \frac{\partial E}{\partial f_t} \frac{\partial E}{\partial o_t} \frac{\partial E}{\partial g_t}]$ 3. $(1 \times 1 \times N)$ the gradient w.r.t. the sequence continuation indicators δ_t is currently not computed.

Implements caffe::Layer< Dtype >.

5.61.2.3 ExactNumBottomBlobs()

```
template<typename Dtype >
virtual int caffe::LSTMUnitLayer< Dtype >::ExactNumBottomBlobs ( ) const [inline], [virtual]
```

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.61.2.4 ExactNumTopBlobs()

```
template<typename Dtype >
virtual int caffe::LSTMUnitLayer< Dtype >::ExactNumTopBlobs ( ) const [inline], [virtual]
```

Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.61.2.5 Forward_cpu()

```
template<typename Dtype >
void caffe::LSTMUnitLayer< Dtype >::Forward_cpu (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [protected], [virtual]
```

Parameters

bottom	input Blob vector (length 3) 1. $(1 \times N \times D)$ the previous timestep cell state c_{t-1} 2. $(1 \times N \times 4D)$ the "gate inputs" $[i'_t, f'_t, o'_t, g'_t]$ 3. $(1 \times N)$ the sequence continuation indicators δ_t
top	output Blob vector (length 2) 1. $(1 \times N \times D)$ the updated cell state c_t, computed as: i_t := [i_t'] f_t := [f_t'] o_t := [o_t'] g_t := [g_t'] c_t := cont_t $*$ (f_t .$*$ c_{t-1}) + (i_t .$*$ g_t) 2. $(1 \times N \times D)$ the updated hidden state h_t, computed as: h_t := o_t .$*$ [c_t]

Implements caffe::Layer< Dtype >.

5.61.2.6 Reshape()

```
template<typename Dtype >
void caffe::LSTMUnitLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

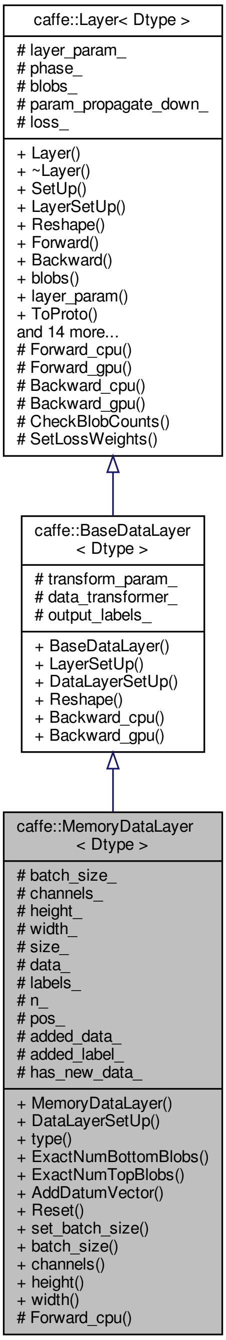
- include/caffe/layers/lstm_layer.hpp
- src/caffe/layers/lstm_unit_layer.cpp

5.62 caffe::MemoryDataLayer< Dtype > Class Template Reference

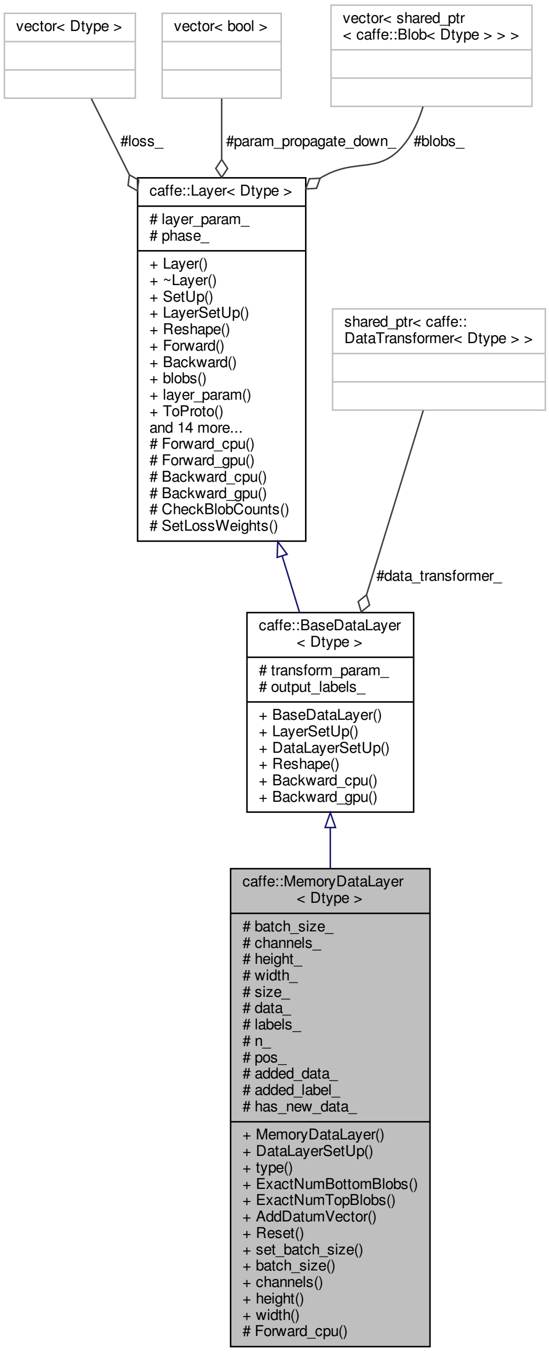
Provides data to the Net from memory.

```
#include <memory_data_layer.hpp>
```

Inheritance diagram for caffe::MemoryDataLayer< Dtype >:



Collaboration diagram for caffe::MemoryDataLayer< Dtype >:



Public Member Functions

- **MemoryDataLayer** (const LayerParameter &param)
- virtual void **DataLayerSetUp** (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)
- virtual const char ∗ type () const

    Returns the layer type.

- virtual int ExactNumBottomBlobs () const

  Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.
- virtual int ExactNumTopBlobs () const

  Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.
- virtual void **AddDatumVector** (const vector< Datum > &datum_vector)
- void **Reset** (Dtype ∗data, Dtype ∗label, int n)
- void **set_batch_size** (int new_size)
- int **batch_size** ()
- int **channels** ()
- int **height** ()
- int **width** ()

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Using the CPU device, compute the layer output.

Protected Attributes

- int **batch_size_**
- int **channels_**
- int **height_**
- int **width_**
- int **size_**
- Dtype ∗ **data_**
- Dtype ∗ **labels_**
- int **n_**
- size_t **pos_**
- Blob< Dtype > **added_data_**
- Blob< Dtype > **added_label_**
- bool **has_new_data_**

5.62.1 Detailed Description

template<typename Dtype>
class caffe::MemoryDataLayer< Dtype >

Provides data to the Net from memory.

TODO(dox): thorough documentation for Forward and proto params.

5.62.2 Member Function Documentation

5.62.2.1 ExactNumBottomBlobs()

```
template<typename Dtype >
virtual int caffe::MemoryDataLayer< Dtype >::ExactNumBottomBlobs ( ) const [inline], [virtual]
```

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.62.2.2 ExactNumTopBlobs()

```
template<typename Dtype >
virtual int caffe::MemoryDataLayer< Dtype >::ExactNumTopBlobs ( ) const [inline], [virtual]
```

Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

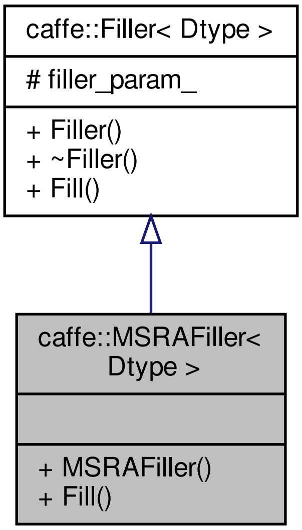
- include/caffe/layers/memory_data_layer.hpp
- src/caffe/layers/memory_data_layer.cpp

5.63 caffe::MSRAFiller< Dtype > Class Template Reference

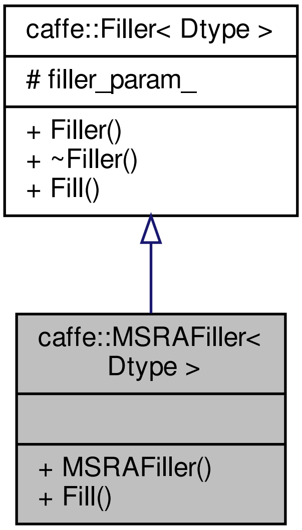
Fills a Blob with values $x \sim N(0, \sigma^2)$ where σ^2 is set inversely proportional to number of incoming nodes, outgoing nodes, or their average.

```
#include <filler.hpp>
```

Inheritance diagram for caffe::MSRAFiller< Dtype >:



Collaboration diagram for caffe::MSRAFiller< Dtype >:



Public Member Functions

- **MSRAFiller** (const FillerParameter &param)
- virtual void **Fill** (Blob< Dtype > ∗blob)

Additional Inherited Members

5.63.1 Detailed Description

template<typename Dtype>
class caffe::MSRAFiller< Dtype >

Fills a Blob with values $x \sim N(0, \sigma^2)$ where σ^2 is set inversely proportional to number of incoming nodes, outgoing nodes, or their average.

A Filler based on the paper [He, Zhang, Ren and Sun 2015]: Specifically accounts for ReLU nonlinearities.

Aside: for another perspective on the scaling factor, see the derivation of [Saxe, McClelland, and Ganguli 2013 (v3)].

It fills the incoming matrix by randomly sampling Gaussian data with std = sqrt(2 / n) where n is the fan_in, fan_out, or their average, depending on the variance_norm option. You should make sure the input blob has shape (num, a, b, c) where a $*$ b $*$ c = fan_in and num $*$ b $*$ c = fan_out. Note that this is currently not the case for inner product layers.

The documentation for this class was generated from the following file:

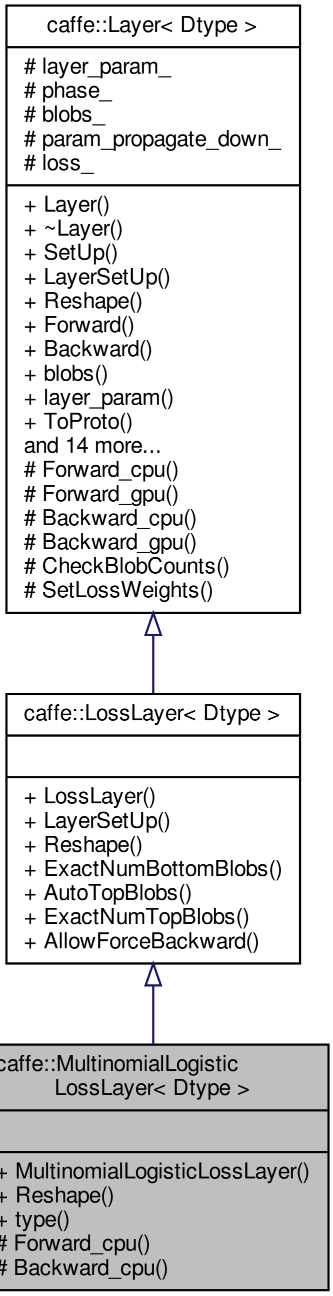
- include/caffe/filler.hpp

5.64 caffe::MultinomialLogisticLossLayer< Dtype > Class Template Reference

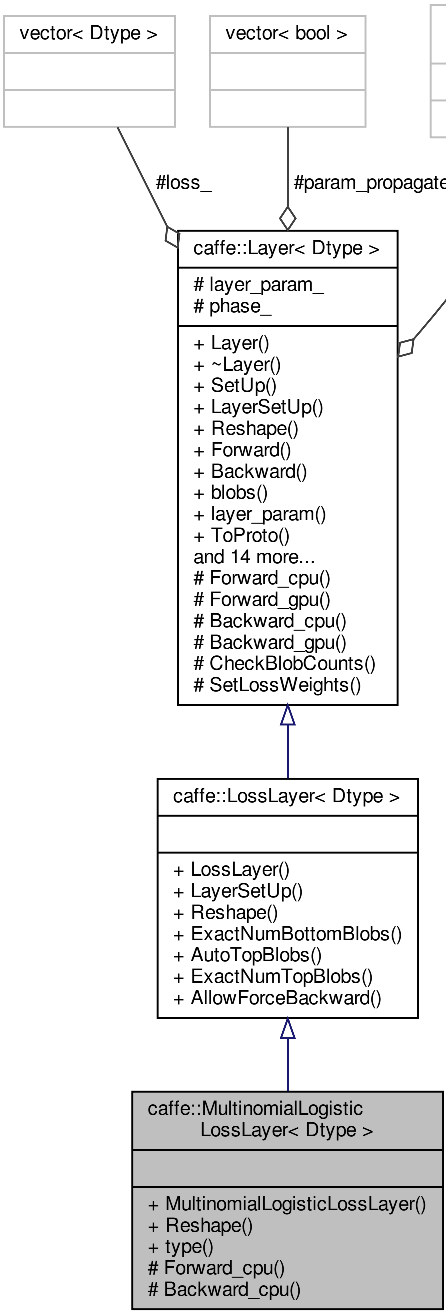
Computes the multinomial logistic loss for a one-of-many classification task, directly taking a predicted probability distribution as input.

`#include <multinomial_logistic_loss_layer.hpp>`

Inheritance diagram for caffe::MultinomialLogisticLossLayer< Dtype >:



Collaboration diagram for caffe::MultinomialLogisticLossLayer< Dtype >:



Public Member Functions

- **MultinomialLogisticLossLayer** (const LayerParameter &param)
- virtual void Reshape (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

    Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.
- virtual const char ∗ type () const

    Returns the layer type.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Computes the multinomial logistic loss for a one-of-many classification task, directly taking a predicted probability distribution as input.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Computes the multinomial logistic loss error gradient w.r.t. the predictions.

Additional Inherited Members

5.64.1 Detailed Description

template<typename Dtype>
class caffe::MultinomialLogisticLossLayer< Dtype >

Computes the multinomial logistic loss for a one-of-many classification task, directly taking a predicted probability distribution as input.

When predictions are not already a probability distribution, you should instead use the SoftmaxWithLossLayer, which maps predictions to a distribution using the SoftmaxLayer, before computing the multinomial logistic loss. The SoftmaxWithLossLayer should be preferred over separate SoftmaxLayer + MultinomialLogisticLossLayer as its gradient computation is more numerically stable.

Parameters

bottom	input Blob vector (length 2) 1. $(N \times C \times H \times W)$ the predictions $\hat{p}$, a Blob with values in $[0, 1]$ indicating the predicted probability of each of the $K = CHW$ classes. Each prediction vector $\hat{p}_n$ should sum to 1 as in a probability distribution: $\forall n \sum\limits_{k=1}^{K} \hat{p}_{nk} = 1$. 2. $(N \times 1 \times 1 \times 1)$ the labels l, an integer-valued Blob with values $l_n \in [0, 1, 2, ..., K - 1]$ indicating the correct class label among the K classes
top	output Blob vector (length 1) 1. $(1 \times 1 \times 1 \times 1)$ the computed multinomial logistic loss: $E = \frac{-1}{N} \sum\limits_{n=1}^{N} \log(\hat{p}_{n,l_n})$

5.64.2 Member Function Documentation

5.64.2.1 Backward_cpu()

```
template<typename Dtype >
void caffe::MultinomialLogisticLossLayer< Dtype >::Backward_cpu (
```

```
const vector< Blob< Dtype > *> & top,
const vector< bool > & propagate_down,
const vector< Blob< Dtype > *> & bottom ) [protected], [virtual]
```

Computes the multinomial logistic loss error gradient w.r.t. the predictions.

Gradients cannot be computed with respect to the label inputs (bottom[1]), so this method ignores bottom[1] and requires !propagate_down[1], crashing if propagate_down[1] is set.

Parameters

top	output Blob vector (length 1), providing the error gradient with respect to the outputs 1. $(1 \times 1 \times 1 \times 1)$ This Blob's diff will simply contain the loss_weight$*$ λ, as λ is the coefficient of this layer's output ℓ_i in the overall Net loss $E = \lambda_i \ell_i +$ other loss terms; hence $\frac{\partial E}{\partial \ell_i} = \lambda_i$. ($*$Assuming that this top Blob is not used as a bottom (input) by any other layer of the Net.)
propagate_down	see Layer::Backward. propagate_down[1] must be false as we can't compute gradients with respect to the labels.
bottom	input Blob vector (length 2) 1. $(N \times C \times H \times W)$ the predictions $\hat{p}$; Backward computes diff $\frac{\partial E}{\partial \hat{p}}$ 2. $(N \times 1 \times 1 \times 1)$ the labels – ignored as we can't compute their error gradients

Implements caffe::Layer$<$ Dtype $>$.

5.64.2.2 Forward_cpu()

```
template<typename Dtype >
void caffe::MultinomialLogisticLossLayer< Dtype >::Forward_cpu (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [protected], [virtual]
```

Computes the multinomial logistic loss for a one-of-many classification task, directly taking a predicted probability distribution as input.

When predictions are not already a probability distribution, you should instead use the SoftmaxWithLossLayer, which maps predictions to a distribution using the SoftmaxLayer, before computing the multinomial logistic loss. The SoftmaxWithLossLayer should be preferred over separate SoftmaxLayer + MultinomialLogisticLossLayer as its gradient computation is more numerically stable.

Parameters

bottom	input Blob vector (length 2) 1. $(N \times C \times H \times W)$ the predictions $\hat{p}$, a Blob with values in $[0, 1]$ indicating the predicted probability of each of the $K = CHW$ classes. Each prediction vector $\hat{p}_n$ should sum to 1 as in a probability distribution: $\forall n \sum\limits_{k=1}^{K} \hat{p}_{nk} = 1$. 2. $(N \times 1 \times 1 \times 1)$ the labels l, an integer-valued Blob with values $l_n \in [0, 1, 2, ..., K-1]$ indicating the correct class label among the K classes
top	output Blob vector (length 1) 1. $(1 \times 1 \times 1 \times 1)$ the computed multinomial logistic loss: $E = \frac{-1}{N} \sum\limits_{n=1}^{N} \log(\hat{p}_{n,l_n})$

Implements caffe::Layer< Dtype >.

5.64.2.3 Reshape()

```
template<typename Dtype >
void caffe::MultinomialLogisticLossLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Reimplemented from caffe::LossLayer< Dtype >.

The documentation for this class was generated from the following files:

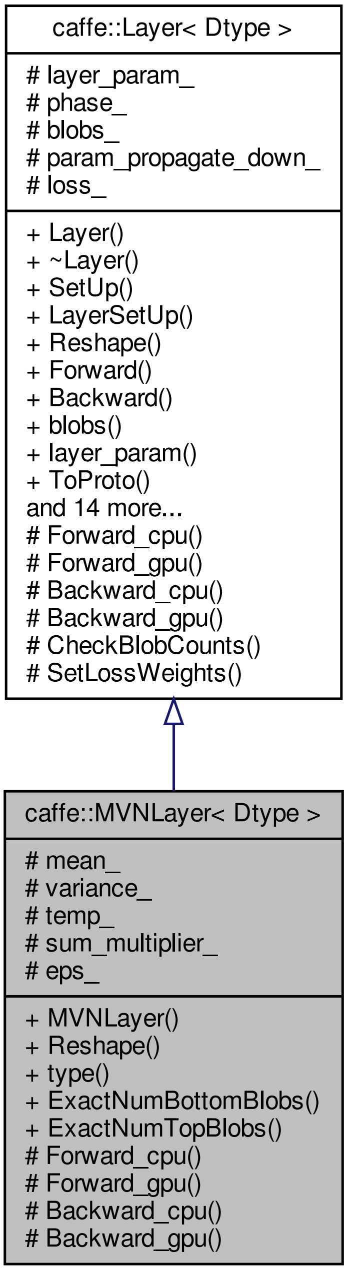
- include/caffe/layers/multinomial_logistic_loss_layer.hpp
- src/caffe/layers/multinomial_logistic_loss_layer.cpp

5.65 caffe::MVNLayer< Dtype > Class Template Reference

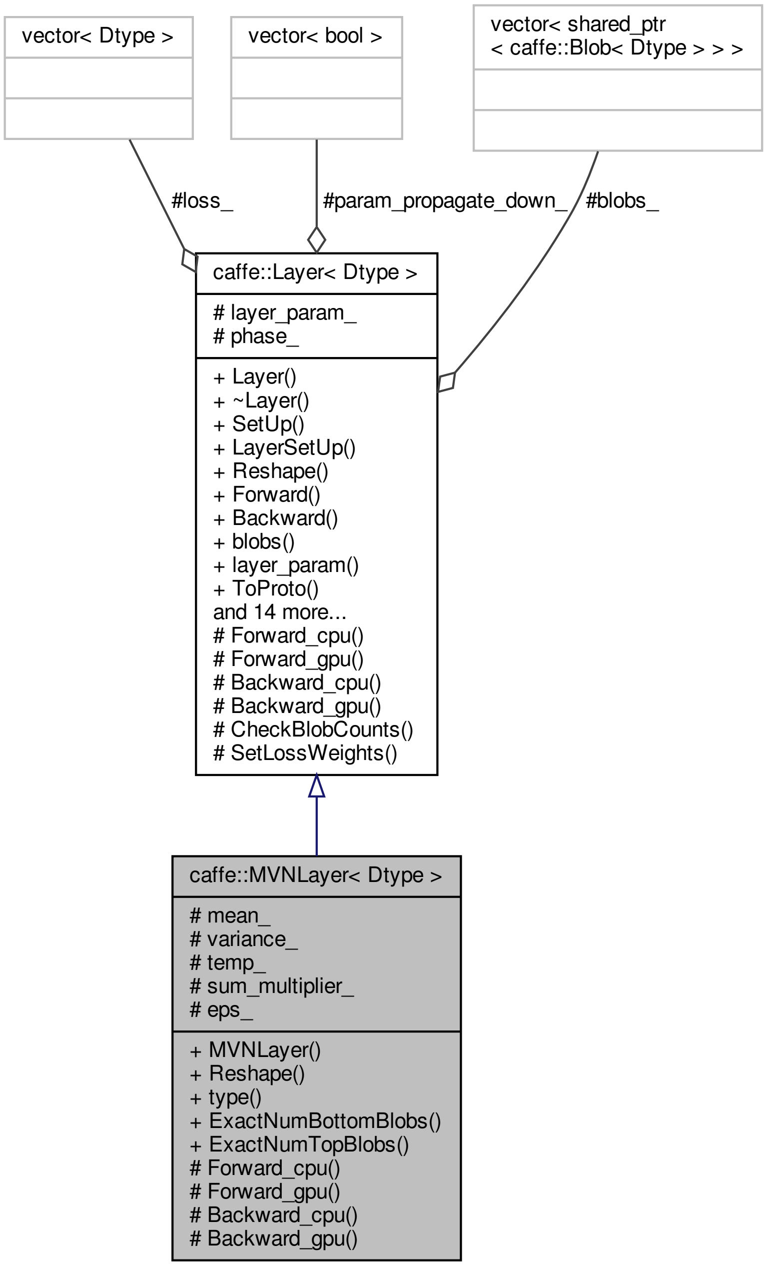
Normalizes the input to have 0-mean and/or unit (1) variance.

```
#include <mvn_layer.hpp>
```

Inheritance diagram for caffe::MVNLayer< Dtype >:



Collaboration diagram for caffe::MVNLayer< Dtype >:



Public Member Functions

- **MVNLayer** (const LayerParameter &param)
- virtual void Reshape (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.
- virtual const char * type () const

  Returns the layer type.
- virtual int ExactNumBottomBlobs () const

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

- virtual int ExactNumTopBlobs () const

  Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the CPU device, compute the layer output.
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the CPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Protected Attributes

- Blob< Dtype > **mean_**
- Blob< Dtype > **variance_**
- Blob< Dtype > **temp_**
- Blob< Dtype > sum_multiplier_

  sum_multiplier is used to carry out sum using BLAS
- Dtype **eps_**

5.65.1 Detailed Description

template<typename Dtype>
class caffe::MVNLayer< Dtype >

Normalizes the input to have 0-mean and/or unit (1) variance.

TODO(dox): thorough documentation for Forward, Backward, and proto params.

5.65.2 Member Function Documentation

5.65.2.1 ExactNumBottomBlobs()

```
template<typename Dtype >
virtual int caffe::MVNLayer< Dtype >::ExactNumBottomBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.65.2.2 ExactNumTopBlobs()

```
template<typename Dtype >
virtual int caffe::MVNLayer< Dtype >::ExactNumTopBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.65.2.3 Reshape()

```
template<typename Dtype >
void caffe::MVNLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

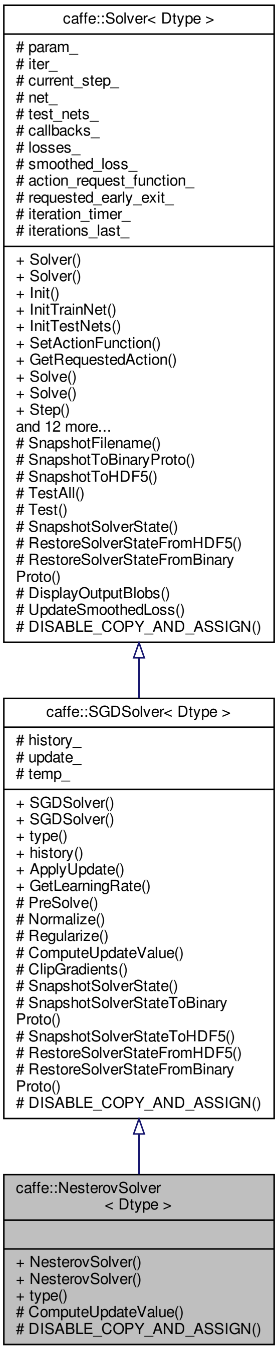
Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

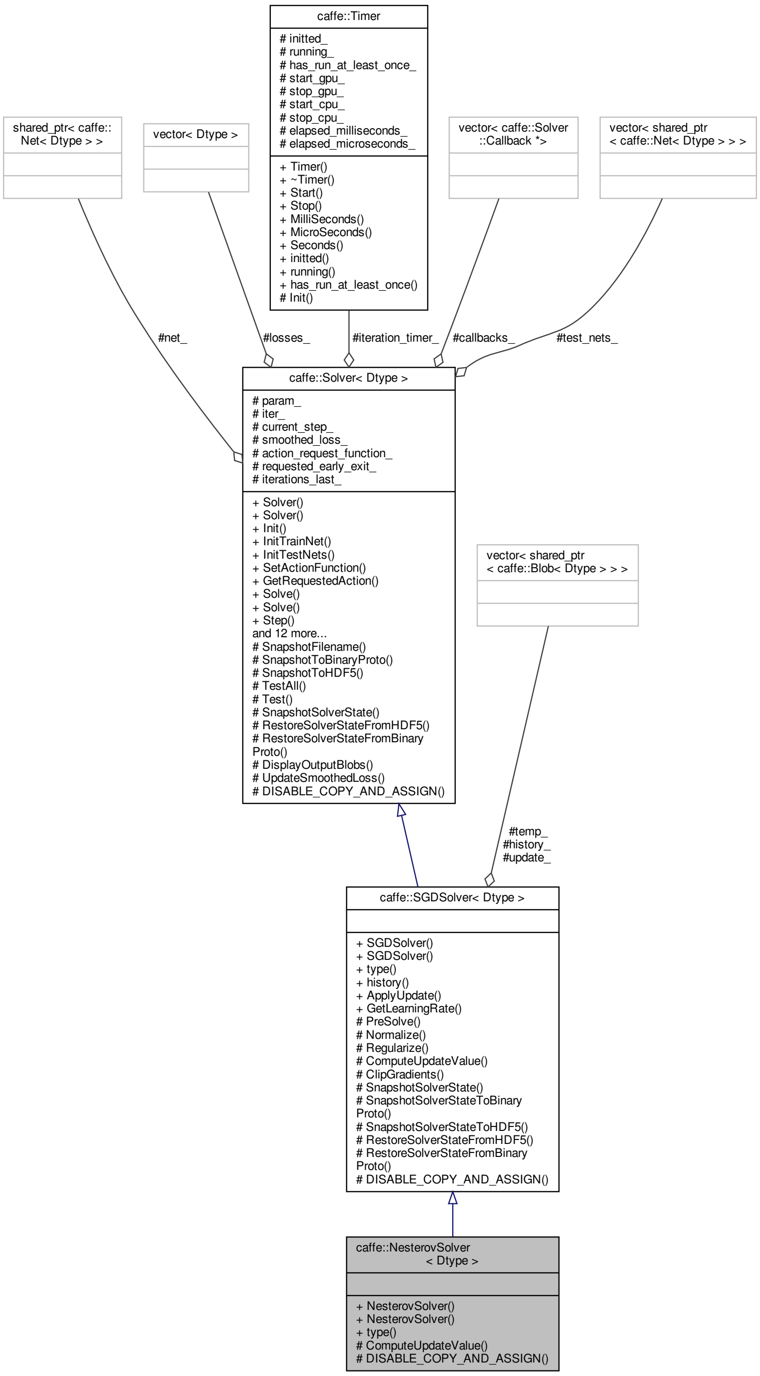
- include/caffe/layers/mvn_layer.hpp
- src/caffe/layers/mvn_layer.cpp

5.66 caffe::NesterovSolver< Dtype > Class Template Reference

Inheritance diagram for caffe::NesterovSolver< Dtype >:



Collaboration diagram for caffe::NesterovSolver< Dtype >:



Public Member Functions

- **NesterovSolver** (const SolverParameter &param)
- **NesterovSolver** (const string &param_file)
- virtual const char $*$ type () const

  Returns the solver type.

Protected Member Functions

- virtual void **ComputeUpdateValue** (int param_id, Dtype rate)
- **DISABLE_COPY_AND_ASSIGN** (NesterovSolver)

Additional Inherited Members

The documentation for this class was generated from the following files:

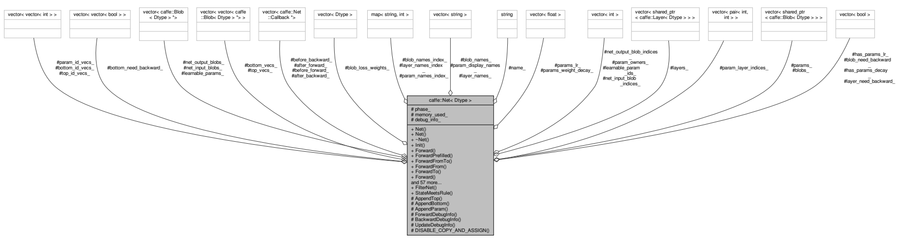
- include/caffe/sgd_solvers.hpp
- src/caffe/solvers/nesterov_solver.cpp

5.67 caffe::Net< Dtype > Class Template Reference

Connects Layers together into a directed acyclic graph (DAG) specified by a NetParameter.

`#include <net.hpp>`

Collaboration diagram for caffe::Net< Dtype >:



Classes

- class Callback

Public Member Functions

- **Net** (const NetParameter &param)
- **Net** (const string &param_file, Phase phase, const int level=0, const vector< string > *stages=NULL)
- void Init (const NetParameter &param)

  Initialize a network with a NetParameter.
- const vector< Blob< Dtype > * > & Forward (Dtype *loss=NULL)

  Run Forward and return the result.
- const vector< Blob< Dtype > * > & ForwardPrefilled (Dtype *loss=NULL)

  DEPRECATED; use Forward() instead.
- Dtype ForwardFromTo (int start, int end)
- Dtype **ForwardFrom** (int start)
- Dtype **ForwardTo** (int end)
- const vector< Blob< Dtype > * > & Forward (const vector< Blob< Dtype > * > &bottom, Dtype *loss=N←ULL)

DEPRECATED; set input blobs then use Forward() instead.

- void ClearParamDiffs ()

  Zeroes out the diffs of all net parameters. Should be run before Backward.
- void Backward ()
- void **BackwardFromTo** (int start, int end)
- void **BackwardFrom** (int start)
- void **BackwardTo** (int end)
- void Reshape ()

  Reshape all layers from bottom to top.
- Dtype **ForwardBackward** ()
- void Update ()

  Updates the network weights based on the diff values computed.
- void ShareWeights ()

  Shares weight data of owner blobs with shared blobs.
- void ShareTrainedLayersWith (const Net *other)

  For an already initialized net, implicitly copies (i.e., using no additional memory) the pre-trained layers from another Net.
- void CopyTrainedLayersFrom (const NetParameter &param)

  For an already initialized net, copies the pre-trained layers from another Net.
- void **CopyTrainedLayersFrom** (const string &trained_filename)
- void **CopyTrainedLayersFromBinaryProto** (const string &trained_filename)
- void **CopyTrainedLayersFromHDF5** (const string &trained_filename)
- void ToProto (NetParameter *param, bool write_diff=false) const

  Writes the net to a proto.
- void ToHDF5 (const string &filename, bool write_diff=false) const

  Writes the net to an HDF5 file.
- const string & name () const

  returns the network name.
- const vector< string > & layer_names () const

  returns the layer names
- const vector< string > & blob_names () const

  returns the blob names
- const vector< shared_ptr< Blob< Dtype > > > & blobs () const

  returns the blobs
- const vector< shared_ptr< Layer< Dtype > > > & layers () const

  returns the layers
- Phase phase () const

  returns the phase: TRAIN or TEST
- const vector< vector< Blob< Dtype > * > > & bottom_vecs () const

  returns the bottom vecs for each layer – usually you won't need this unless you do per-layer checks such as gradients.
- const vector< vector< Blob< Dtype > * > > & top_vecs () const

  returns the top vecs for each layer – usually you won't need this unless you do per-layer checks such as gradients.
- const vector< int > & top_ids (int i) const

  returns the ids of the top blobs of layer i
- const vector< int > & bottom_ids (int i) const

  returns the ids of the bottom blobs of layer i
- const vector< vector< bool > > & **bottom_need_backward** () const
- const vector< Dtype > & **blob_loss_weights** () const
- const vector< bool > & **layer_need_backward** () const
- const vector< shared_ptr< Blob< Dtype > > > & params () const

  returns the parameters
- const vector< Blob< Dtype > * > & **learnable_params** () const

- const vector< float > & params_lr () const

  returns the learnable parameter learning rate multipliers
- const vector< bool > & **has_params_lr** () const
- const vector< float > & params_weight_decay () const

  returns the learnable parameter decay multipliers
- const vector< bool > & **has_params_decay** () const
- const map< string, int > & **param_names_index** () const
- const vector< int > & **param_owners** () const
- const vector< string > & **param_display_names** () const
- int num_inputs () const

  Input and output blob numbers.
- int **num_outputs** () const
- const vector< Blob< Dtype > * > & **input_blobs** () const
- const vector< Blob< Dtype > * > & **output_blobs** () const
- const vector< int > & **input_blob_indices** () const
- const vector< int > & **output_blob_indices** () const
- bool **has_blob** (const string &blob_name) const
- const shared_ptr< Blob< Dtype > > **blob_by_name** (const string &blob_name) const
- bool **has_layer** (const string &layer_name) const
- const shared_ptr< Layer< Dtype > > **layer_by_name** (const string &layer_name) const
- void **set_debug_info** (const bool value)
- const vector< Callback * > & **before_forward** () const
- void **add_before_forward** (Callback *value)
- const vector< Callback * > & **after_forward** () const
- void **add_after_forward** (Callback *value)
- const vector< Callback * > & **before_backward** () const
- void **add_before_backward** (Callback *value)
- const vector< Callback * > & **after_backward** () const
- void **add_after_backward** (Callback *value)

Static Public Member Functions

- static void FilterNet (const NetParameter &param, NetParameter *param_filtered)

  Remove layers that the user specified should be excluded given the current phase, level, and stage.
- static bool StateMeetsRule (const NetState &state, const NetStateRule &rule, const string &layer_name)

  return whether NetState state meets NetStateRule rule

Protected Member Functions

- void AppendTop (const NetParameter &param, const int layer_id, const int top_id, set< string > *available_blobs, map< string, int > *blob_name_to_idx)

  Append a new top blob to the net.
- int AppendBottom (const NetParameter &param, const int layer_id, const int bottom_id, set< string > *available_blobs, map< string, int > *blob_name_to_idx)

  Append a new bottom blob to the net.
- void AppendParam (const NetParameter &param, const int layer_id, const int param_id)

  Append a new parameter blob to the net.
- void ForwardDebugInfo (const int layer_id)

  Helper for displaying debug info in Forward.
- void BackwardDebugInfo (const int layer_id)

  Helper for displaying debug info in Backward.
- void UpdateDebugInfo (const int param_id)

  Helper for displaying debug info in Update.
- **DISABLE_COPY_AND_ASSIGN** (Net)

Protected Attributes

- string name_

  The network name.
- Phase phase_

  The phase: TRAIN or TEST.
- vector< shared_ptr< Layer< Dtype > > > layers_

  Individual layers in the net.
- vector< string > **layer_names_**
- map< string, int > **layer_names_index_**
- vector< bool > **layer_need_backward_**
- vector< shared_ptr< Blob< Dtype > > > blobs_

  the blobs storing intermediate results between the layer.
- vector< string > **blob_names_**
- map< string, int > **blob_names_index_**
- vector< bool > **blob_need_backward_**
- vector< vector< Blob< Dtype > * > > bottom_vecs_
- vector< vector< int > > **bottom_id_vecs_**
- vector< vector< bool > > **bottom_need_backward_**
- vector< vector< Blob< Dtype > * > > top_vecs_

  top_vecs stores the vectors containing the output for each layer
- vector< vector< int > > **top_id_vecs_**
- vector< Dtype > blob_loss_weights_
- vector< vector< int > > **param_id_vecs_**
- vector< int > **param_owners_**
- vector< string > **param_display_names_**
- vector< pair< int, int > > **param_layer_indices_**
- map< string, int > **param_names_index_**
- vector< int > net_input_blob_indices_

  blob indices for the input and the output of the net
- vector< int > **net_output_blob_indices_**
- vector< Blob< Dtype > * > **net_input_blobs_**
- vector< Blob< Dtype > * > **net_output_blobs_**
- vector< shared_ptr< Blob< Dtype > > > params_

  The parameters in the network.
- vector< Blob< Dtype > * > **learnable_params_**
- vector< int > learnable_param_ids_
- vector< float > params_lr_

  *the learning rate multipliers for learnable_params_*
- vector< bool > **has_params_lr_**
- vector< float > params_weight_decay_

  *the weight decay multipliers for learnable_params_*
- vector< bool > **has_params_decay_**
- size_t memory_used_

  The bytes of memory used by this net.
- bool debug_info_

  Whether to compute and display debug info for the net.
- vector< Callback * > **before_forward_**
- vector< Callback * > **after_forward_**
- vector< Callback * > **before_backward_**
- vector< Callback * > **after_backward_**

5.67.1 Detailed Description

template<typename Dtype>
class caffe::Net< Dtype >

Connects Layers together into a directed acyclic graph (DAG) specified by a NetParameter.

TODO(dox): more thorough description.

5.67.2 Member Function Documentation

5.67.2.1 Backward()

```
template<typename Dtype >
void caffe::Net< Dtype >::Backward ( )
```

The network backward should take no input and output, since it solely computes the gradient w.r.t the parameters, and the data has already been provided during the forward pass.

5.67.2.2 ForwardFromTo()

```
template<typename Dtype >
Dtype caffe::Net< Dtype >::ForwardFromTo (
             int start,
             int end )
```

The From and To variants of Forward and Backward operate on the (topological) ordering by which the net is specified. For general DAG networks, note that (1) computing from one layer to another might entail extra computation on unrelated branches, and (2) computation starting in the middle may be incorrect if all of the layers of a fan-in are not included.

5.67.2.3 Reshape()

```
template<typename Dtype >
void caffe::Net< Dtype >::Reshape ( )
```

Reshape all layers from bottom to top.

This is useful to propagate changes to layer sizes without running a forward pass, e.g. to compute output feature size.

5.67.2.4 ShareWeights()

```
template<typename Dtype >
void caffe::Net< Dtype >::ShareWeights ( )
```

Shares weight data of owner blobs with shared blobs.

Note: this is called by Net::Init, and thus should normally not be called manually.

5.67.3 Member Data Documentation

5.67.3.1 blob_loss_weights_

```
template<typename Dtype >
vector<Dtype> caffe::Net< Dtype >::blob_loss_weights_  [protected]
```

Vector of weight in the loss (or objective) function of each net blob, indexed by blob_id.

5.67.3.2 bottom_vecs_

```
template<typename Dtype >
vector<vector<Blob<Dtype>*> > caffe::Net< Dtype >::bottom_vecs_  [protected]
```

bottom_vecs stores the vectors containing the input for each layer. They don't actually host the blobs (blobs_ does), so we simply store pointers.

5.67.3.3 learnable_param_ids_

```
template<typename Dtype >
vector<int> caffe::Net< Dtype >::learnable_param_ids_  [protected]
```

The mapping from params_ -> learnable_params_: we have learnable_param_ids_.size() == params_.size(), and learnable_params_[learnable_param_ids_[i]] == params_[i].get() if and only if params_[i] is an "owner"; otherwise, params_[i] is a sharer and learnable_params_[learnable_param_ids_[i]] gives its owner.

The documentation for this class was generated from the following files:

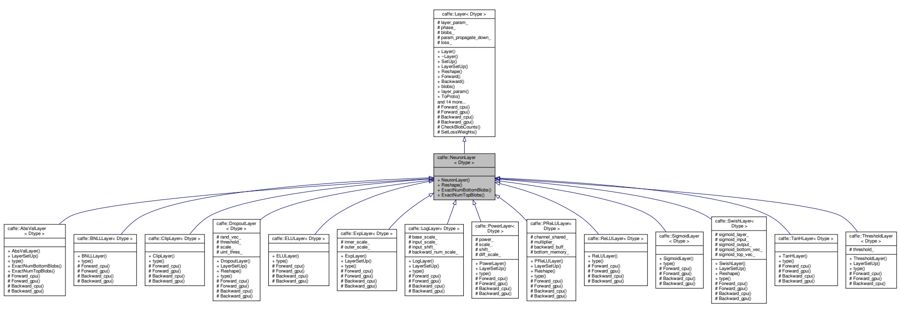
- include/caffe/net.hpp
- src/caffe/net.cpp

5.68 caffe::NeuronLayer< Dtype > Class Template Reference

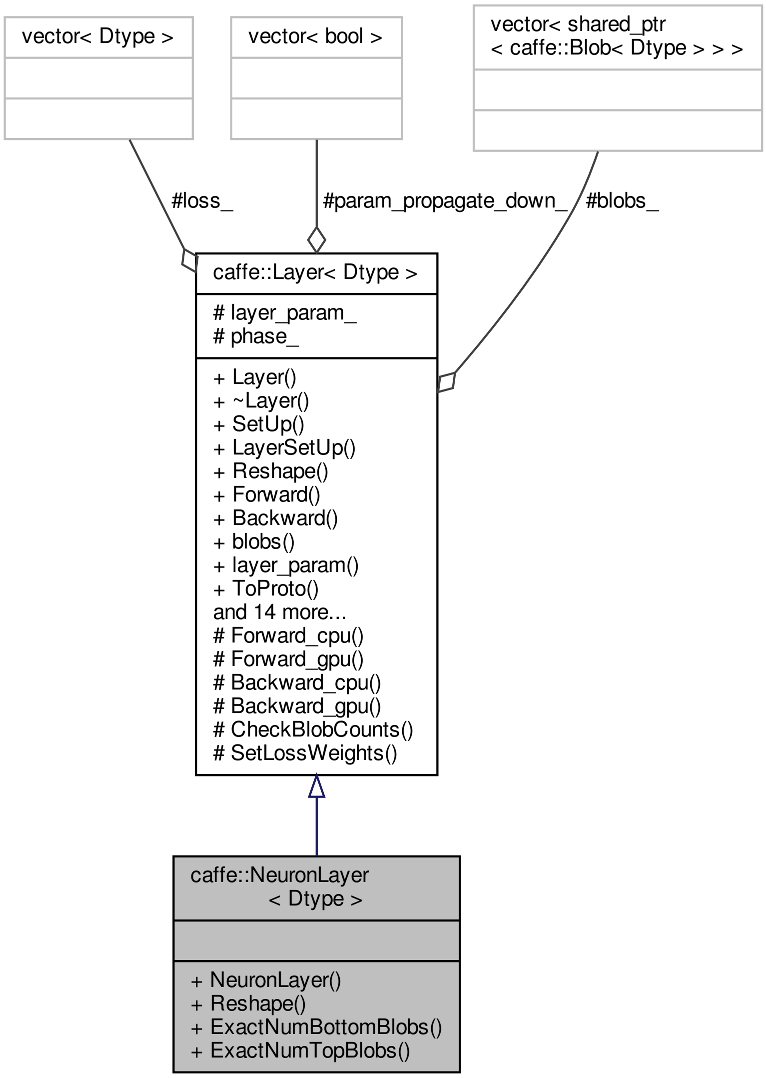
An interface for layers that take one blob as input (x) and produce one equally-sized blob as output (y), where each element of the output depends only on the corresponding input element.

`#include <neuron_layer.hpp>`

Inheritance diagram for caffe::NeuronLayer< Dtype >:



Collaboration diagram for caffe::NeuronLayer< Dtype >:



Public Member Functions

- **NeuronLayer** (const LayerParameter &param)
- virtual void Reshape (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.
- virtual int ExactNumBottomBlobs () const

  Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.
- virtual int ExactNumTopBlobs () const

  Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

Additional Inherited Members

5.68.1 Detailed Description

template<typename Dtype>
class caffe::NeuronLayer< Dtype >

An interface for layers that take one blob as input (x) and produce one equally-sized blob as output (y), where each element of the output depends only on the corresponding input element.

5.68.2 Member Function Documentation

5.68.2.1 ExactNumBottomBlobs()

```
template<typename Dtype >
virtual int caffe::NeuronLayer< Dtype >::ExactNumBottomBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

Reimplemented in caffe::AbsValLayer< Dtype >.

5.68.2.2 ExactNumTopBlobs()

```
template<typename Dtype >
virtual int caffe::NeuronLayer< Dtype >::ExactNumTopBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

Reimplemented in caffe::AbsValLayer< Dtype >.

5.68.2.3 Reshape()

```
template<typename Dtype >
void caffe::NeuronLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Implements caffe::Layer< Dtype >.

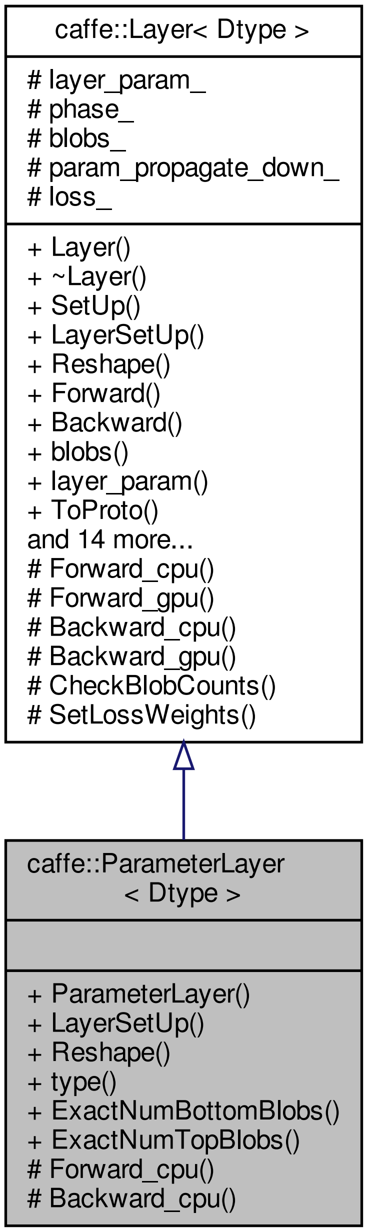
Reimplemented in caffe::PReLULayer< Dtype >, caffe::DropoutLayer< Dtype >, and caffe::SwishLayer< Dtype >.

The documentation for this class was generated from the following files:

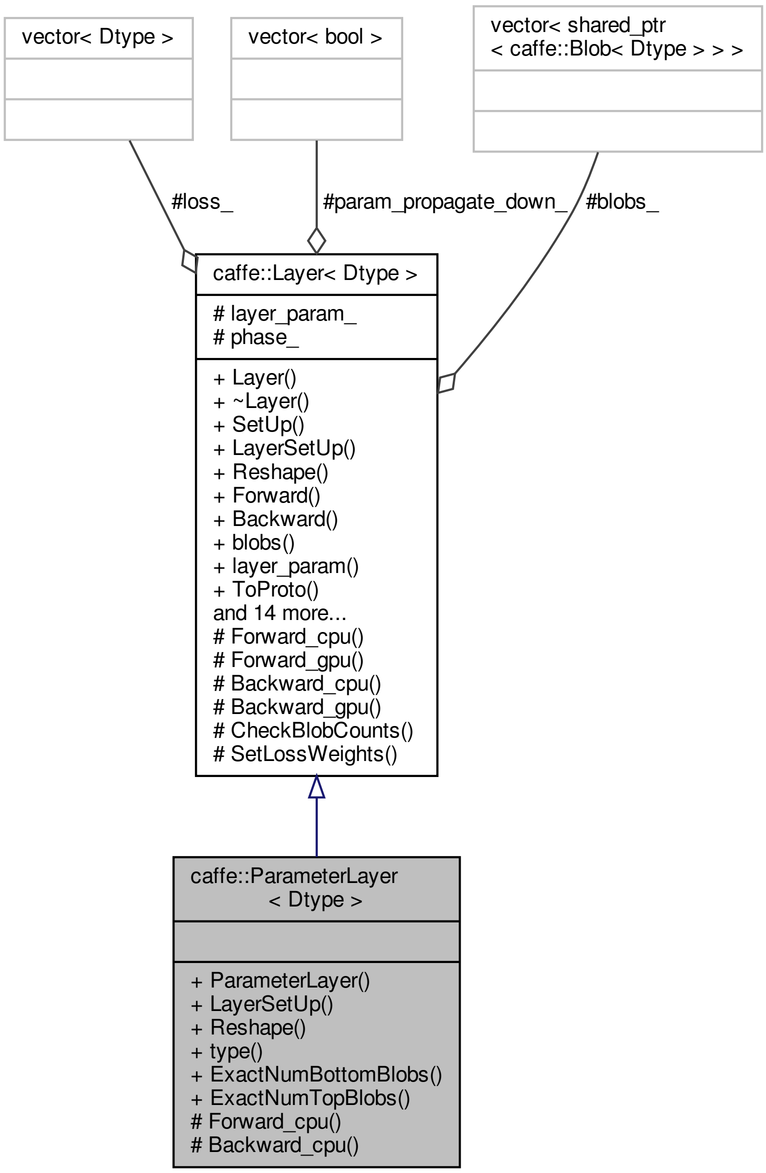
- include/caffe/layers/neuron_layer.hpp
- src/caffe/layers/neuron_layer.cpp

5.69 caffe::ParameterLayer< Dtype > Class Template Reference

Inheritance diagram for caffe::ParameterLayer< Dtype >:



Collaboration diagram for caffe::ParameterLayer< Dtype >:



Public Member Functions

- **ParameterLayer** (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void Reshape (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.
- virtual const char ∗ type () const

  Returns the layer type.
- virtual int ExactNumBottomBlobs () const

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

- virtual int ExactNumTopBlobs () const

  Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the CPU device, compute the layer output.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the CPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true.

Additional Inherited Members

5.69.1 Member Function Documentation

5.69.1.1 ExactNumBottomBlobs()

```
template<typename Dtype >
virtual int caffe::ParameterLayer< Dtype >::ExactNumBottomBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.69.1.2 ExactNumTopBlobs()

```
template<typename Dtype >
virtual int caffe::ParameterLayer< Dtype >::ExactNumTopBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.69.1.3 LayerSetUp()

```
template<typename Dtype >
virtual void caffe::ParameterLayer< Dtype >::LayerSetUp (
             const vector< Blob< Dtype > *> & bottom,
             const vector< Blob< Dtype > *> & top )  [inline], [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

5.69.1.4 Reshape()

```
template<typename Dtype >
virtual void caffe::ParameterLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [inline], [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following file:

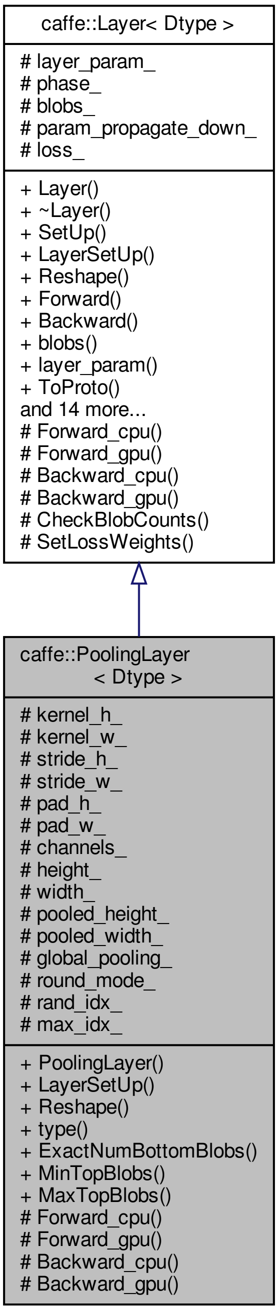
- include/caffe/layers/parameter_layer.hpp

5.70 caffe::PoolingLayer< Dtype > Class Template Reference

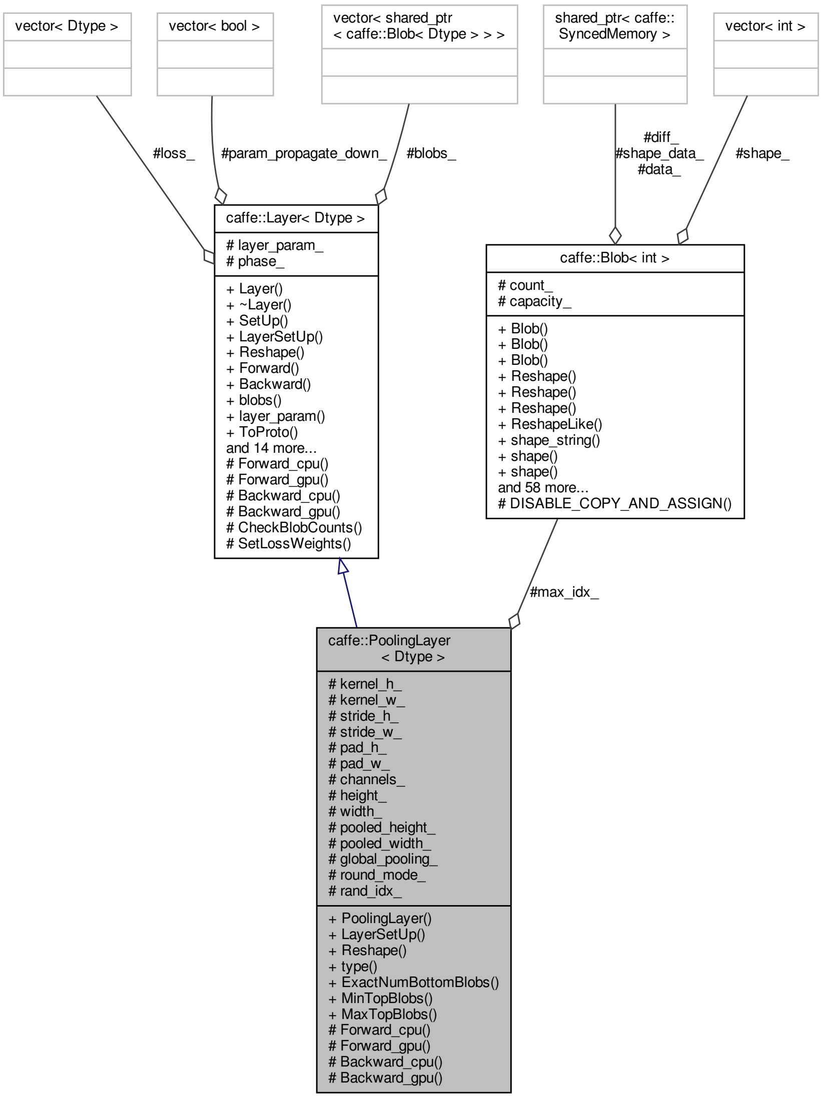
Pools the input image by taking the max, average, etc. within regions.

`#include <pooling_layer.hpp>`

Inheritance diagram for caffe::PoolingLayer< Dtype >:



Collaboration diagram for caffe::PoolingLayer< Dtype >:



Public Member Functions

- **PoolingLayer** (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void Reshape (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.
- virtual const char ∗ type () const

  Returns the layer type.
- virtual int ExactNumBottomBlobs () const

  Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.
- virtual int MinTopBlobs () const

  Returns the minimum number of top blobs required by the layer, or -1 if no minimum number is required.
- virtual int MaxTopBlobs () const

  Returns the maximum number of top blobs required by the layer, or -1 if no maximum number is required.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the CPU device, compute the layer output.
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the CPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Protected Attributes

- int **kernel_h_**
- int **kernel_w_**
- int **stride_h_**
- int **stride_w_**
- int **pad_h_**
- int **pad_w_**
- int **channels_**
- int **height_**
- int **width_**
- int **pooled_height_**
- int **pooled_width_**
- bool **global_pooling_**
- PoolingParameter_RoundMode **round_mode_**
- Blob< Dtype > **rand_idx_**
- Blob< int > **max_idx_**

5.70.1 Detailed Description

template<typename Dtype>
class caffe::PoolingLayer< Dtype >

Pools the input image by taking the max, average, etc. within regions.

TODO(dox): thorough documentation for Forward, Backward, and proto params.

5.70.2 Member Function Documentation

5.70.2.1 ExactNumBottomBlobs()

```
template<typename Dtype >
virtual int caffe::PoolingLayer< Dtype >::ExactNumBottomBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.70.2.2 LayerSetUp()

```
template<typename Dtype >
void caffe::PoolingLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

5.70.2.3 MaxTopBlobs()

```
template<typename Dtype >
virtual int caffe::PoolingLayer< Dtype >::MaxTopBlobs ( ) const  [inline], [virtual]
```

Returns the maximum number of top blobs required by the layer, or -1 if no maximum number is required.

This method should be overridden to return a non-negative value if your layer expects some maximum number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.70.2.4 MinTopBlobs()

```
template<typename Dtype >
virtual int caffe::PoolingLayer< Dtype >::MinTopBlobs ( ) const  [inline], [virtual]
```

Returns the minimum number of top blobs required by the layer, or -1 if no minimum number is required.

This method should be overridden to return a non-negative value if your layer expects some minimum number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.70.2.5 Reshape()

```
template<typename Dtype >
void caffe::PoolingLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

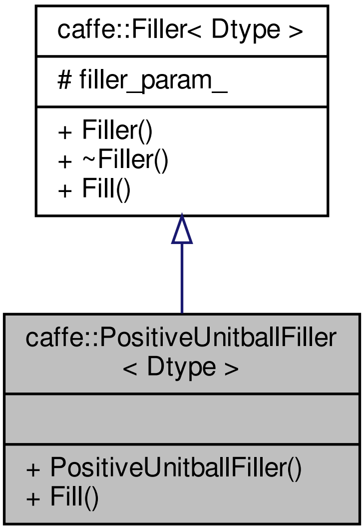
- include/caffe/layers/pooling_layer.hpp
- src/caffe/layers/pooling_layer.cpp

5.71 caffe::PositiveUnitballFiller< Dtype > Class Template Reference

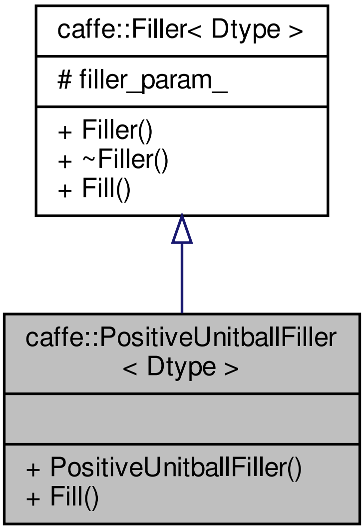
Fills a Blob with values $x \in [0, 1]$ such that $\forall i \sum_j x_{ij} = 1$.

```
#include <filler.hpp>
```

Inheritance diagram for caffe::PositiveUnitballFiller< Dtype >:



Collaboration diagram for caffe::PositiveUnitballFiller< Dtype >:



Public Member Functions

- **PositiveUnitballFiller** (const FillerParameter &param)
- virtual void **Fill** (Blob< Dtype > ∗blob)

Additional Inherited Members

5.71.1 Detailed Description

template<typename Dtype>
class caffe::PositiveUnitballFiller< Dtype >

Fills a Blob with values $x \in [0, 1]$ such that $\forall i \sum_j x_{ij} = 1$.

The documentation for this class was generated from the following file:

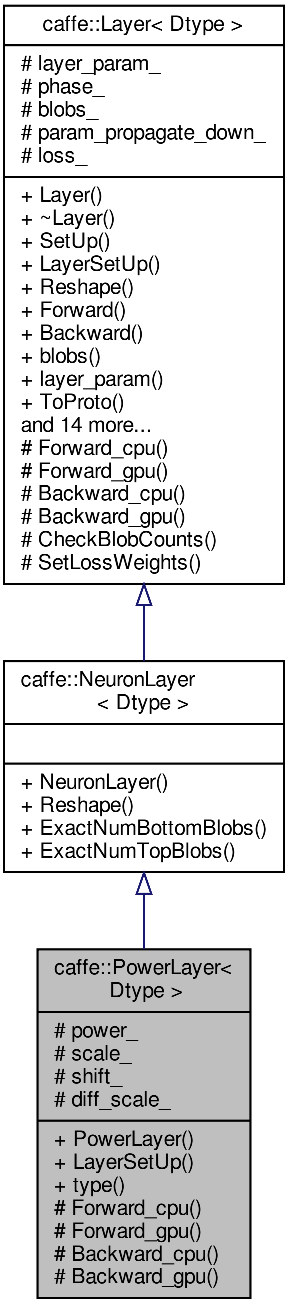
- include/caffe/filler.hpp

5.72 caffe::PowerLayer< Dtype > Class Template Reference

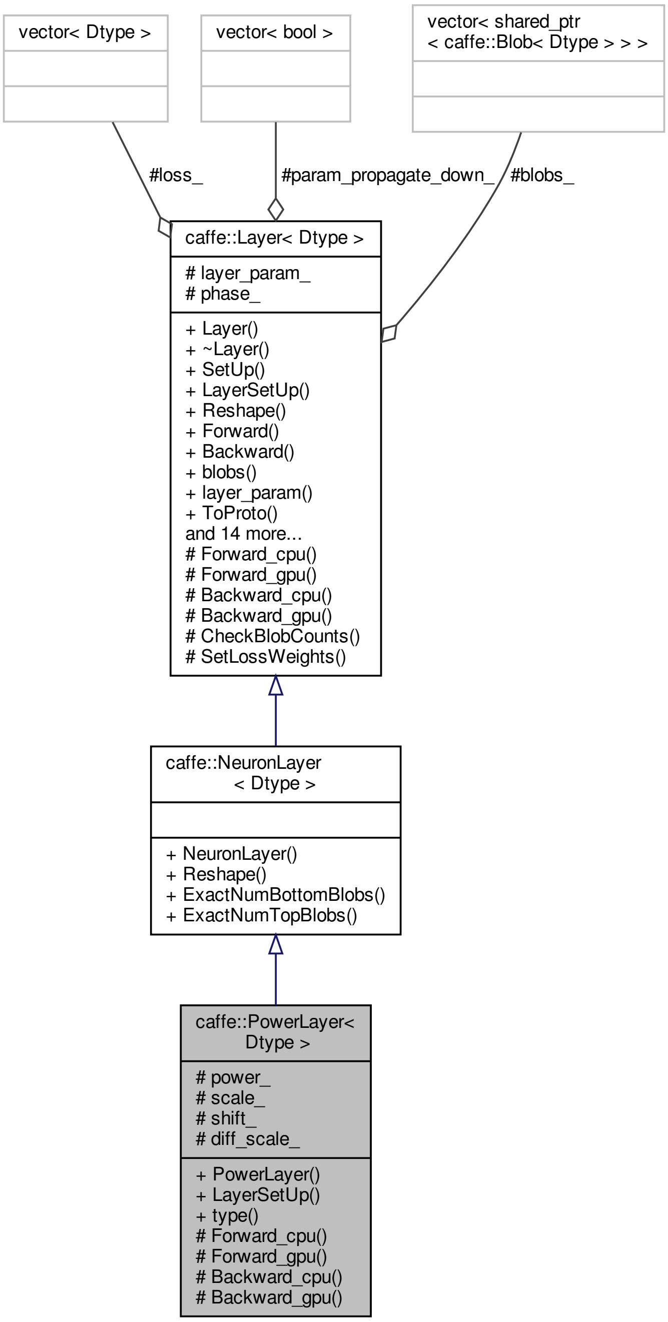
Computes $y = (\alpha x + \beta)^\gamma$, as specified by the scale α, shift β, and power γ.

`#include <power_layer.hpp>`

Inheritance diagram for caffe::PowerLayer< Dtype >:



Collaboration diagram for caffe::PowerLayer< Dtype >:



Public Member Functions

- PowerLayer (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual const char ∗ type () const

  Returns the layer type.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

    Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

    Computes the error gradient w.r.t. the power inputs.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

    Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Protected Attributes

- Dtype power_

    γ from layer_param_.power_param()
- Dtype scale_

    α from layer_param_.power_param()
- Dtype shift_

    β from layer_param_.power_param()
- Dtype diff_scale_

    Result of $\alpha\gamma$.

5.72.1 Detailed Description

template<typename Dtype>
class caffe::PowerLayer< Dtype >

Computes $y = (\alpha x + \beta)^\gamma$, as specified by the scale α, shift β, and power γ.

5.72.2 Constructor & Destructor Documentation

5.72.2.1 PowerLayer()

```
template<typename Dtype >
caffe::PowerLayer< Dtype >::PowerLayer (
            const LayerParameter & param )  [inline], [explicit]
```

Parameters

param	provides PowerParameter power_param, with PowerLayer options: • scale (**optional**, default 1) the scale α • shift (**optional**, default 0) the shift β • power (**optional**, default 1) the power γ

5.72.3 Member Function Documentation

5.72.3.1 Backward_cpu()

```
template<typename Dtype >
void caffe::PowerLayer< Dtype >::Backward_cpu (
            const vector< Blob< Dtype > *> & top,
            const vector< bool > & propagate_down,
            const vector< Blob< Dtype > *> & bottom ) [protected], [virtual]
```

Computes the error gradient w.r.t. the power inputs.

Parameters

top	output Blob vector (length 1), providing the error gradient with respect to the outputs 1. $(N \times C \times H \times W)$ containing error gradients $\frac{\partial E}{\partial y}$ with respect to computed outputs y
propagate_down	see Layer::Backward.
bottom	input Blob vector (length 1) 1. $(N \times C \times H \times W)$ the inputs x; Backward fills their diff with gradients $\frac{\partial E}{\partial x} = \frac{\partial E}{\partial y} \alpha \gamma (\alpha x + \beta)^{\gamma - 1} = \frac{\partial E}{\partial y} \frac{\alpha \gamma y}{\alpha x + \beta}$ if propagate_down[0]

Implements caffe::Layer< Dtype >.

5.72.3.2 Forward_cpu()

```
template<typename Dtype >
void caffe::PowerLayer< Dtype >::Forward_cpu (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [protected], [virtual]
```

Parameters

bottom	input Blob vector (length 1) 1. $(N \times C \times H \times W)$ the inputs x
top	output Blob vector (length 1) 1. $(N \times C \times H \times W)$ the computed outputs $y = (\alpha x + \beta)^\gamma$

Implements caffe::Layer< Dtype >.

5.72.3.3 LayerSetUp()

```
template<typename Dtype >
void caffe::PowerLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

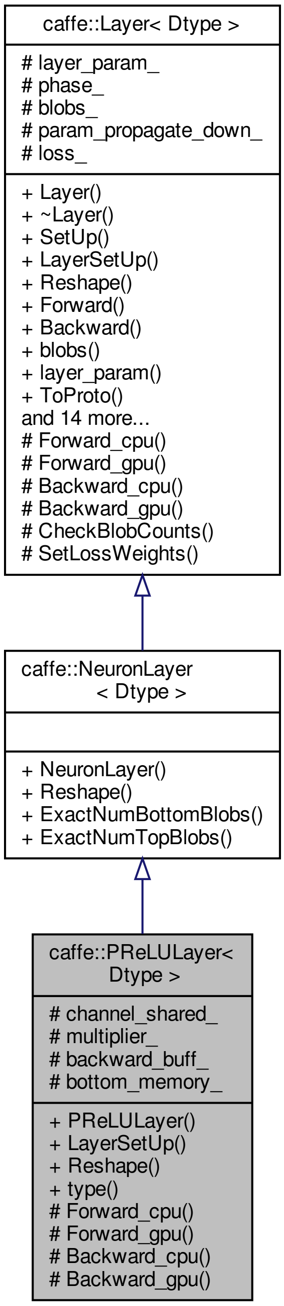
- include/caffe/layers/power_layer.hpp
- src/caffe/layers/power_layer.cpp

5.73 caffe::PReLULayer< Dtype > Class Template Reference

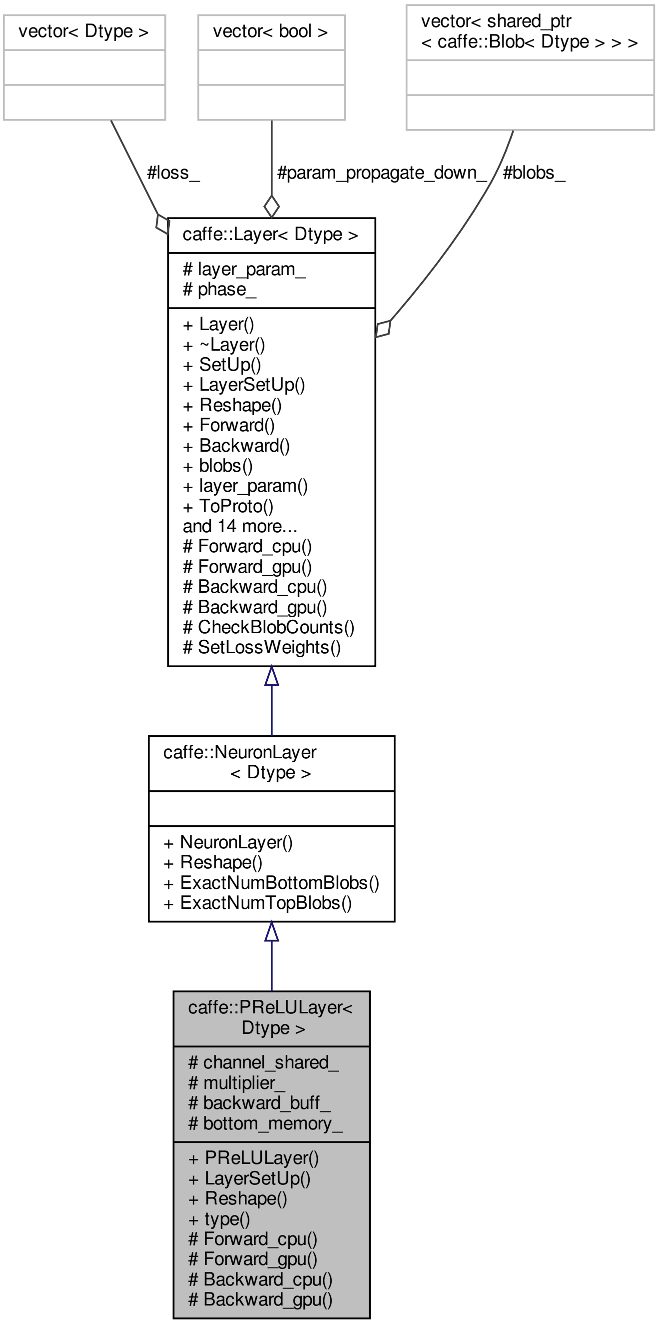
Parameterized Rectified Linear Unit non-linearity $y_i = \max(0, x_i) + a_i \min(0, x_i)$. The differences from ReLU↵ Layer are 1) negative slopes are learnable though backprop and 2) negative slopes can vary across channels. The number of axes of input blob should be greater than or equal to 2. The 1st axis (0-based) is seen as channels.

```
#include <prelu_layer.hpp>
```

Inheritance diagram for caffe::PReLULayer< Dtype >:



Collaboration diagram for caffe::PReLULayer< Dtype >:



Public Member Functions

- PReLULayer (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > $*$> &bottom, const vector< Blob< Dtype > $*$> &top)

  Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void Reshape (const vector< Blob< Dtype > $*$> &bottom, const vector< Blob< Dtype > $*$> &top)

  Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

- virtual const char ∗ type () const

  Returns the layer type.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)
- virtual void Forward_gpu (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > ∗> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > ∗> &bottom)

  Computes the error gradient w.r.t. the PReLU inputs.
- virtual void Backward_gpu (const vector< Blob< Dtype > ∗> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > ∗> &bottom)

  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Protected Attributes

- bool **channel_shared_**
- Blob< Dtype > **multiplier_**
- Blob< Dtype > **backward_buff_**
- Blob< Dtype > **bottom_memory_**

5.73.1 Detailed Description

template<typename Dtype>
class caffe::PReLULayer< Dtype >

Parameterized Rectified Linear Unit non-linearity $y_i = \max(0, x_i) + a_i \min(0, x_i)$. The differences from ReLULayer are 1) negative slopes are learnable though backprop and 2) negative slopes can vary across channels. The number of axes of input blob should be greater than or equal to 2. The 1st axis (0-based) is seen as channels.

5.73.2 Constructor & Destructor Documentation

5.73.2.1 PReLULayer()

```
template<typename Dtype >
caffe::PReLULayer< Dtype >::PReLULayer (
            const LayerParameter & param )  [inline], [explicit]
```

Parameters

param	provides PReLUParameter prelu_param, with PReLULayer options: • filler (**optional**, FillerParameter, default {'type': constant 'value':0.25}). • channel_shared (**optional**, default false). negative slopes are shared across channels.

5.73.3 Member Function Documentation

5.73.3.1 Backward_cpu()

```
template<typename Dtype >
void caffe::PReLULayer< Dtype >::Backward_cpu (
            const vector< Blob< Dtype > *> & top,
            const vector< bool > & propagate_down,
            const vector< Blob< Dtype > *> & bottom )  [protected], [virtual]
```

Computes the error gradient w.r.t. the PReLU inputs.

Parameters

top	output Blob vector (length 1), providing the error gradient with respect to the outputs 1. $(N \times C \times ...)$ containing error gradients $\frac{\partial E}{\partial y}$ with respect to computed outputs y
propagate_down	see Layer::Backward.
bottom	input Blob vector (length 1) 1. $(N \times C \times ...)$ the inputs x; For each channel i, backward fills their diff with gradients $\frac{\partial E}{\partial x_i} = \left\{ \begin{array}{lr} a_i \frac{\partial E}{\partial y_i} & \mathrm{if} \; x_i \le 0 \\ \frac{\partial E}{\partial y_i} & \mathrm{if} \; x_i > 0 \end{array} \right.$. If param_propagate_down_[0] is true, it fills the diff with gradients $\frac{\partial E}{\partial a_i} = \left\{ \begin{array}{lr} \sum_{x_i} x_i \frac{\partial E}{\partial y_i} & \mathrm{if} \; x_i \le 0 \\ 0 & \mathrm{if} \; x_i > 0 \end{array} \right.$.

Implements caffe::Layer< Dtype >.

5.73.3.2 Forward_cpu()

```
template<typename Dtype >
void caffe::PReLULayer< Dtype >::Forward_cpu (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [protected], [virtual]
```

Parameters

bottom	input Blob vector (length 1) 1. $(N \times C \times ...)$ the inputs x
top	output Blob vector (length 1) 1. $(N \times C \times ...)$ the computed outputs for each channel i $y_i = \max(0, x_i) + a_i \min(0, x_i)$.

Implements caffe::Layer< Dtype >.

5.73.3.3 LayerSetUp()

```
template<typename Dtype >
void caffe::PReLULayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

5.73.3.4 Reshape()

```
template<typename Dtype >
void caffe::PReLULayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

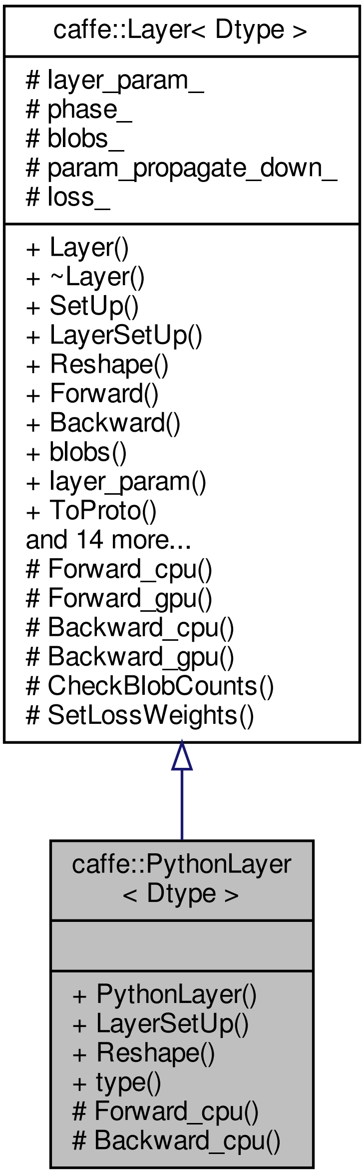
Reimplemented from caffe::NeuronLayer< Dtype >.

The documentation for this class was generated from the following files:

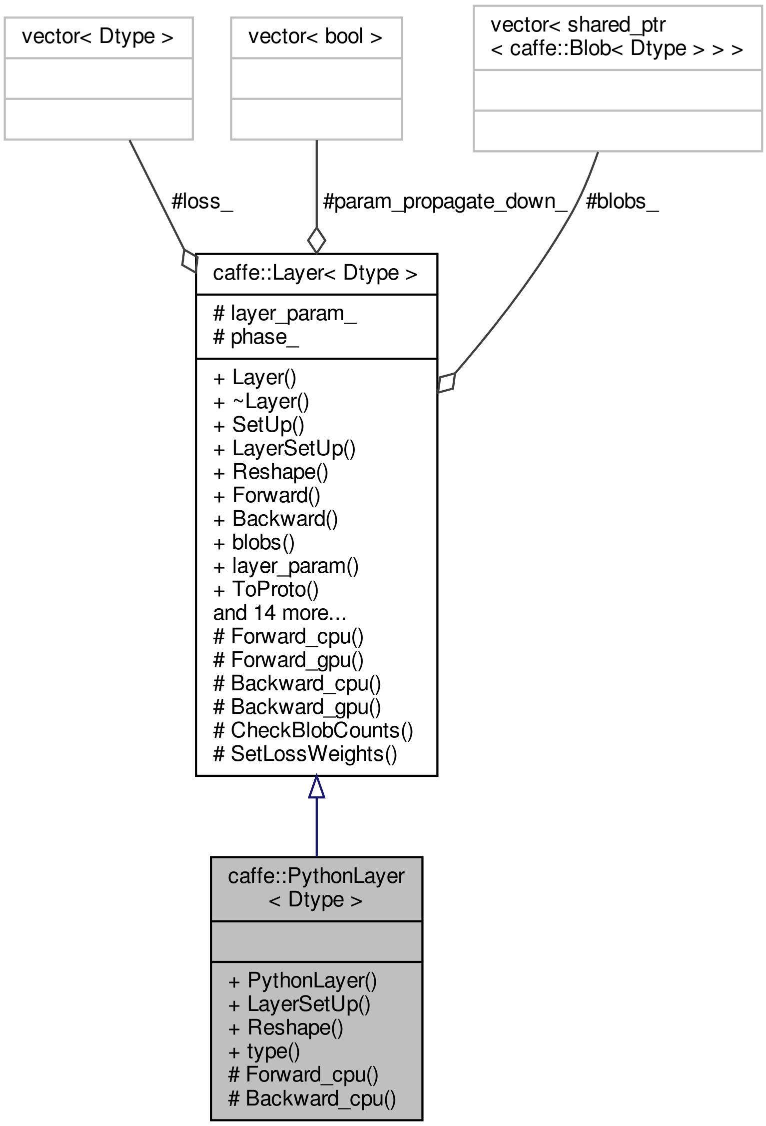
- include/caffe/layers/prelu_layer.hpp
- src/caffe/layers/prelu_layer.cpp

5.74 caffe::PythonLayer< Dtype > Class Template Reference

Inheritance diagram for caffe::PythonLayer< Dtype >:



Collaboration diagram for caffe::PythonLayer< Dtype >:



Public Member Functions

- **PythonLayer** (PyObject ∗self, const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void Reshape (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.
- virtual const char ∗ type () const

  Returns the layer type.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the CPU device, compute the layer output.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the CPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true.

Additional Inherited Members

5.74.1 Member Function Documentation

5.74.1.1 LayerSetUp()

```
template<typename Dtype >
virtual void caffe::PythonLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [inline], [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

5.74.1.2 Reshape()

```
template<typename Dtype >
virtual void caffe::PythonLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [inline], [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following file:

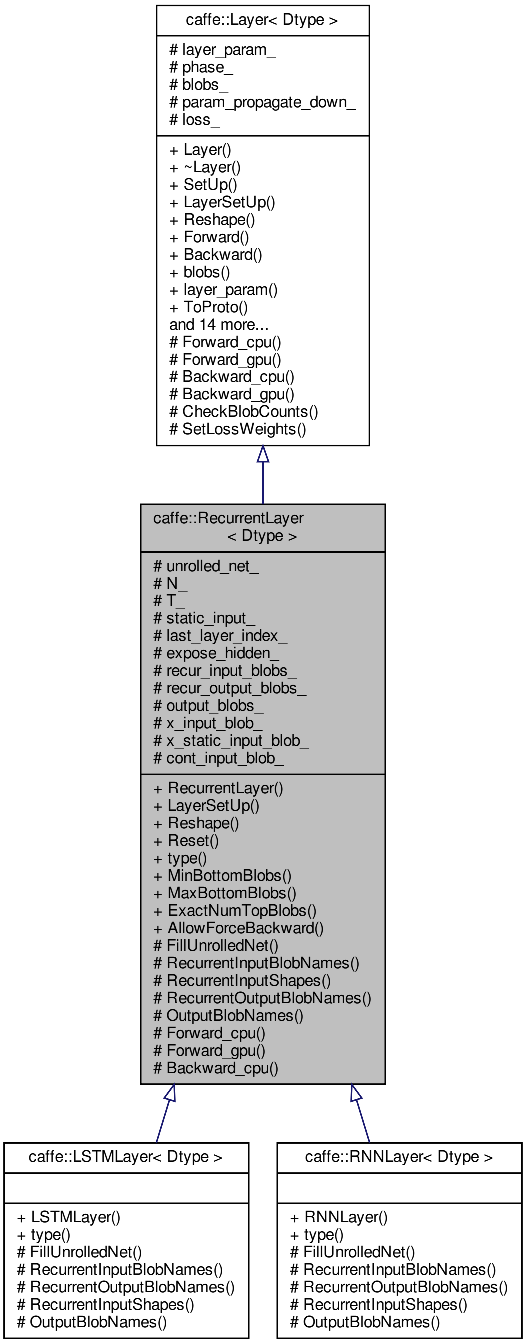
- include/caffe/layers/python_layer.hpp

5.75 caffe::RecurrentLayer< Dtype > Class Template Reference

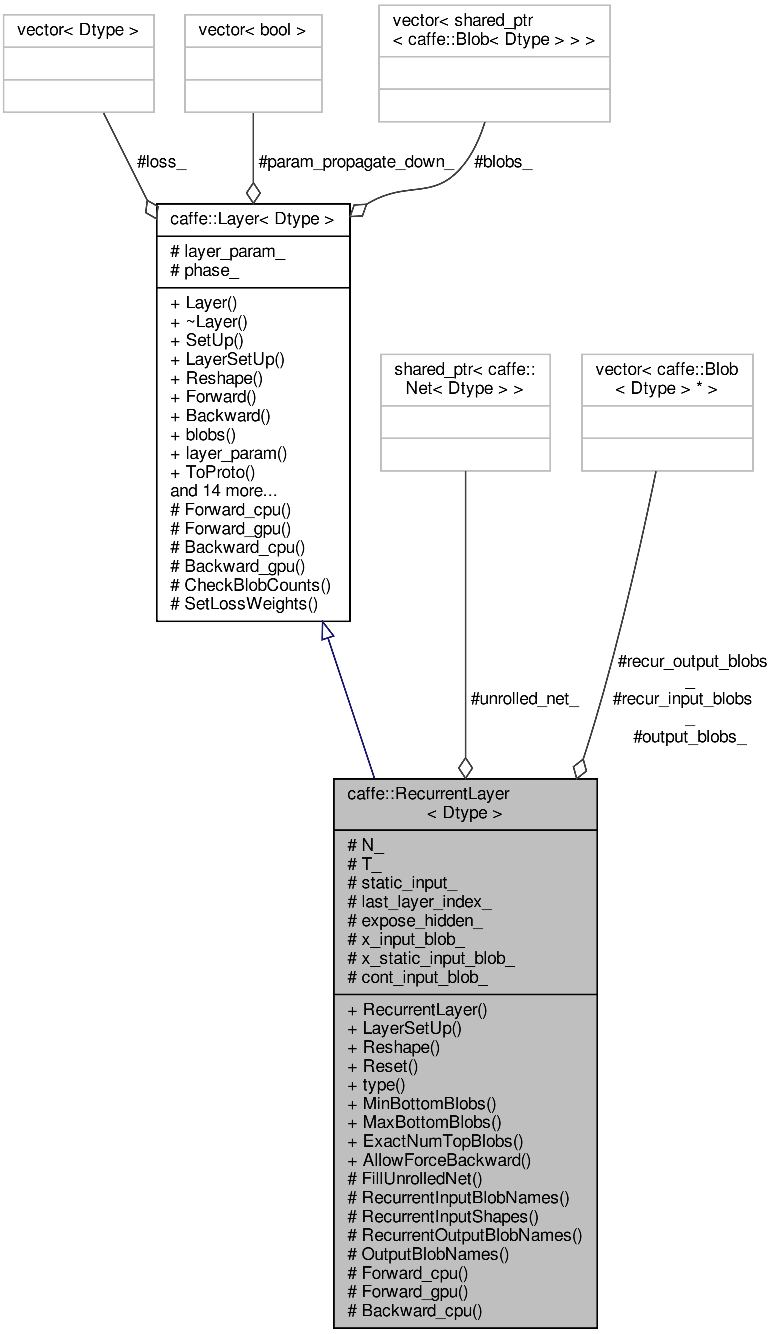
An abstract class for implementing recurrent behavior inside of an unrolled network. This Layer type cannot be instantiated – instead, you should use one of its implementations which defines the recurrent architecture, such as RNNLayer or LSTMLayer.

```
#include <recurrent_layer.hpp>
```

Inheritance diagram for caffe::RecurrentLayer< Dtype >:



Collaboration diagram for caffe::RecurrentLayer< Dtype >:



Public Member Functions

- **RecurrentLayer** (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void Reshape (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

- virtual void **Reset** ()
- virtual const char * type () const

    Returns the layer type.
- virtual int MinBottomBlobs () const

    Returns the minimum number of bottom blobs required by the layer, or -1 if no minimum number is required.
- virtual int MaxBottomBlobs () const

    Returns the maximum number of bottom blobs required by the layer, or -1 if no maximum number is required.
- virtual int ExactNumTopBlobs () const

    Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.
- virtual bool AllowForceBackward (const int bottom_index) const

    Return whether to allow force_backward for a given bottom blob index.

Protected Member Functions

- virtual void FillUnrolledNet (NetParameter *net_param) const =0

    Fills net_param with the recurrent network architecture. Subclasses should define this – see RNNLayer and LSTMLayer for examples.
- virtual void RecurrentInputBlobNames (vector< string > *names) const =0

    Fills names with the names of the 0th timestep recurrent input Blob&s. Subclasses should define this – see RNNLayer and LSTMLayer for examples.
- virtual void RecurrentInputShapes (vector< BlobShape > *shapes) const =0

    Fills shapes with the shapes of the recurrent input Blob&s. Subclasses should define this – see RNNLayer and LSTMLayer for examples.
- virtual void RecurrentOutputBlobNames (vector< string > *names) const =0

    Fills names with the names of the Tth timestep recurrent output Blob&s. Subclasses should define this – see RNNLayer and LSTMLayer for examples.
- virtual void OutputBlobNames (vector< string > *names) const =0

    Fills names with the names of the output blobs, concatenated across all timesteps. Should return a name for each top Blob. Subclasses should define this – see RNNLayer and LSTMLayer for examples.
- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

    Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

    Using the CPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true.

Protected Attributes

- shared_ptr< Net< Dtype > > unrolled_net_

    A Net to implement the Recurrent functionality.
- int N_

    The number of independent streams to process simultaneously.
- int T_

    The number of timesteps in the layer's input, and the number of timesteps over which to backpropagate through time.
- bool static_input_

    Whether the layer has a "static" input copied across all timesteps.
- int last_layer_index_

    The last layer to run in the network. (Any later layers are losses added to force the recurrent net to do backprop.)
- bool expose_hidden_

Whether the layer's hidden state at the first and last timesteps are layer inputs and outputs, respectively.

- vector< Blob< Dtype > *> **recur_input_blobs_**
- vector< Blob< Dtype > *> **recur_output_blobs_**
- vector< Blob< Dtype > *> **output_blobs_**
- Blob< Dtype > * **x_input_blob_**
- Blob< Dtype > * **x_static_input_blob_**
- Blob< Dtype > * **cont_input_blob_**

5.75.1 Detailed Description

template<typename Dtype>
class caffe::RecurrentLayer< Dtype >

An abstract class for implementing recurrent behavior inside of an unrolled network. This Layer type cannot be instantiated – instead, you should use one of its implementations which defines the recurrent architecture, such as RNNLayer or LSTMLayer.

5.75.2 Member Function Documentation

5.75.2.1 AllowForceBackward()

```
template<typename Dtype >
virtual bool caffe::RecurrentLayer< Dtype >::AllowForceBackward (
            const int bottom_index ) const  [inline], [virtual]
```

Return whether to allow force_backward for a given bottom blob index.

If AllowForceBackward(i) == false, we will ignore the force_backward setting and backpropagate to blob i only if it needs gradient information (as is done when force_backward == false).

Reimplemented from caffe::Layer< Dtype >.

5.75.2.2 ExactNumTopBlobs()

```
template<typename Dtype >
virtual int caffe::RecurrentLayer< Dtype >::ExactNumTopBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.75.2.3 Forward_cpu()

```
template<typename Dtype >
void caffe::RecurrentLayer< Dtype >::Forward_cpu (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [protected], [virtual]
```

Parameters

bottom	input Blob vector (length 2-3)

1. $(T \times N \times ...)$ the time-varying input x. After the first two axes, whose dimensions must correspond to the number of timesteps T and the number of independent streams N, respectively, its dimensions may be arbitrary. Note that the ordering of dimensions – $(T \times N \times ...)$, rather than $(N \times T \times ...)$ – means that the N independent input streams must be "interleaved".
2. $(T \times N)$ the sequence continuation indicators δ. These inputs should be binary (0 or 1) indicators, where $\delta_{t,n} = 0$ means that timestep t of stream n is the beginning of a new sequence, and hence the previous hidden state h_{t-1} is multiplied by $\delta_t = 0$ and has no effect on the cell's output at timestep t, and a value of $\delta_{t,n} = 1$ means that timestep t of stream n is a continuation from the previous timestep $t-1$, and the previous hidden state h_{t-1} affects the updated hidden state and output.
3. $(N \times ...)$ (optional) the static (non-time-varying) input x_{static}. After the first axis, whose dimension must be the number of independent streams, its dimensions may be arbitrary. This is mathematically equivalent to using a time-varying input of $x'_t = [x_t; x_{static}]$ – i.e., tiling the static input across the T timesteps and concatenating with the time-varying input. Note that if this input is used, all timesteps in a single batch within a particular one of the N streams must share the same static input, even if the sequence continuation indicators suggest that difference sequences are ending and beginning within a single batch. This may require padding and/or truncation for uniform length.

Parameters

top	output Blob vector (length 1) 1. $(T \times N \times D)$ the time-varying output y, where D is `recurrent_param.num_output()`. Refer to documentation for particular RecurrentLayer implementations (such as RNNLayer and LSTMLayer) for the definition of y.

Implements caffe::Layer< Dtype >.

5.75.2.4 LayerSetUp()

```
template<typename Dtype >
void caffe::RecurrentLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

5.75.2.5 MaxBottomBlobs()

```
template<typename Dtype >
virtual int caffe::RecurrentLayer< Dtype >::MaxBottomBlobs ( ) const [inline], [virtual]
```

Returns the maximum number of bottom blobs required by the layer, or -1 if no maximum number is required.

This method should be overridden to return a non-negative value if your layer expects some maximum number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.75.2.6 MinBottomBlobs()

```
template<typename Dtype >
virtual int caffe::RecurrentLayer< Dtype >::MinBottomBlobs ( ) const [inline], [virtual]
```

Returns the minimum number of bottom blobs required by the layer, or -1 if no minimum number is required.

This method should be overridden to return a non-negative value if your layer expects some minimum number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.75.2.7 Reshape()

```
template<typename Dtype >
void caffe::RecurrentLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

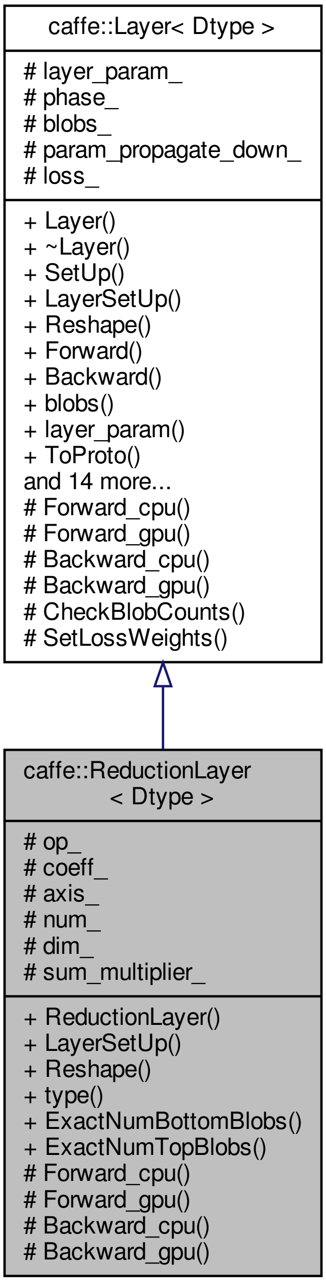
- include/caffe/layers/lstm_layer.hpp
- include/caffe/layers/recurrent_layer.hpp
- src/caffe/layers/recurrent_layer.cpp

5.76 caffe::ReductionLayer< Dtype > Class Template Reference

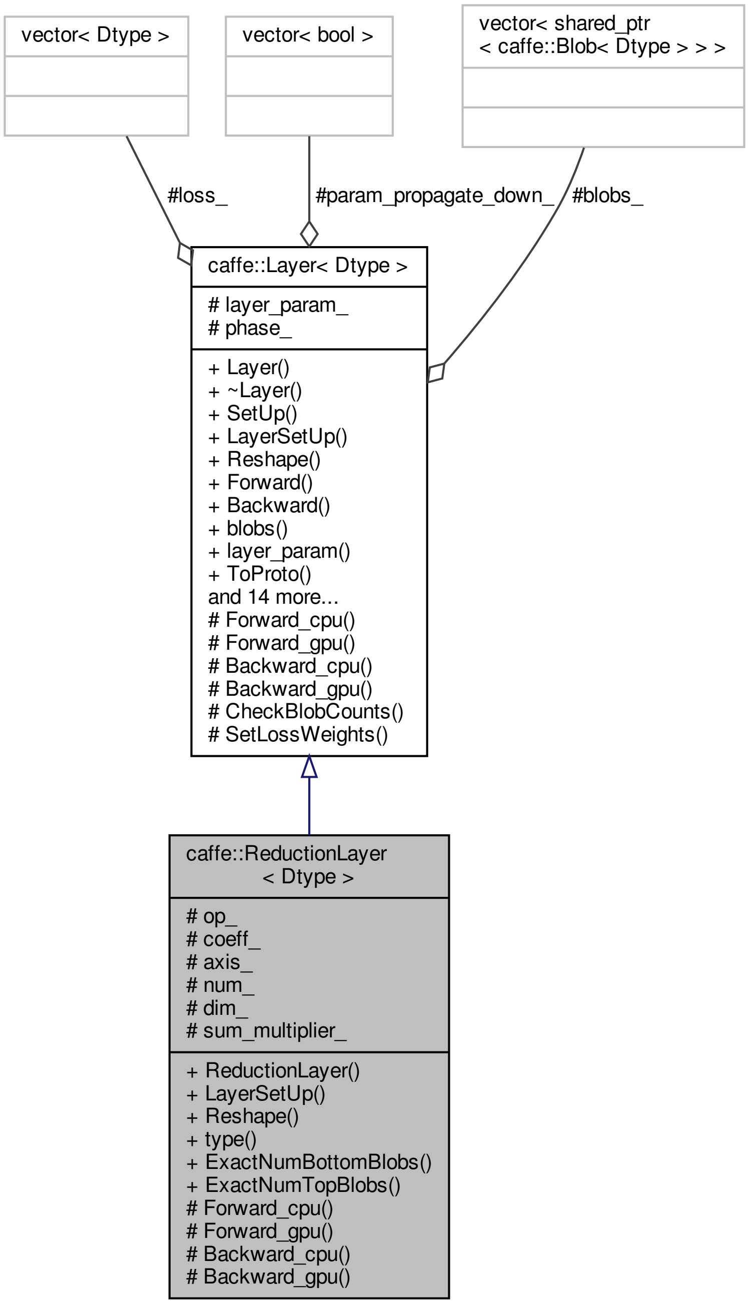
Compute "reductions" – operations that return a scalar output Blob for an input Blob of arbitrary size, such as the sum, absolute sum, and sum of squares.

```
#include <reduction_layer.hpp>
```

Inheritance diagram for caffe::ReductionLayer< Dtype >:



Collaboration diagram for caffe::ReductionLayer< Dtype >:



Public Member Functions

- **ReductionLayer** (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

    Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void Reshape (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

    Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

- virtual const char * type () const

  Returns the layer type.
- virtual int ExactNumBottomBlobs () const

  Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.
- virtual int ExactNumTopBlobs () const

  Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the CPU device, compute the layer output.
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the CPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Protected Attributes

- ReductionParameter_ReductionOp op_

  the reduction operation performed by the layer
- Dtype coeff_

  a scalar coefficient applied to all outputs
- int axis_

  the index of the first input axis to reduce
- int num_

  the number of reductions performed
- int dim_

  the input size of each reduction
- Blob< Dtype > sum_multiplier_

  a helper Blob used for summation (op_ == SUM)

5.76.1 Detailed Description

template<typename Dtype>
class caffe::ReductionLayer< Dtype >

Compute "reductions" – operations that return a scalar output Blob for an input Blob of arbitrary size, such as the sum, absolute sum, and sum of squares.

TODO(dox): thorough documentation for Forward, Backward, and proto params.

5.76.2 Member Function Documentation

5.76.2.1 ExactNumBottomBlobs()

```
template<typename Dtype >
virtual int caffe::ReductionLayer< Dtype >::ExactNumBottomBlobs ( ) const [inline], [virtual]
```

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.76.2.2 ExactNumTopBlobs()

```
template<typename Dtype >
virtual int caffe::ReductionLayer< Dtype >::ExactNumTopBlobs ( ) const [inline], [virtual]
```

Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.76.2.3 LayerSetUp()

```
template<typename Dtype >
void caffe::ReductionLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

5.76.2.4 Reshape()

```
template<typename Dtype >
void caffe::ReductionLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

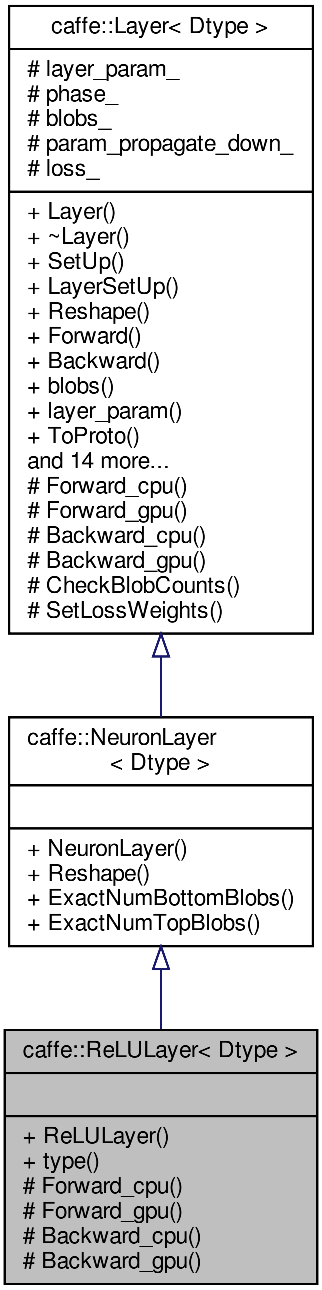
- include/caffe/layers/reduction_layer.hpp
- src/caffe/layers/reduction_layer.cpp

5.77 caffe::ReLULayer< Dtype > Class Template Reference

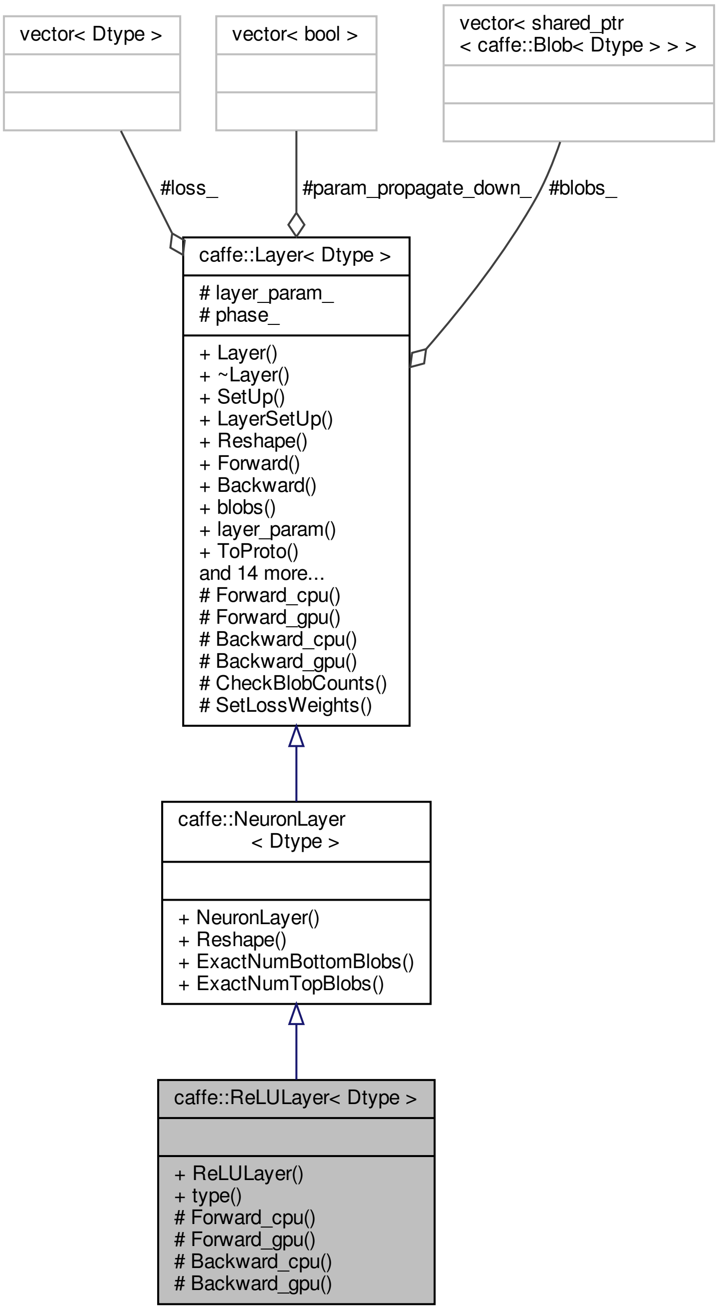
Rectified Linear Unit non-linearity $y = \max(0, x)$. The simple max is fast to compute, and the function does not saturate.

```
#include <relu_layer.hpp>
```

Inheritance diagram for caffe::ReLULayer< Dtype >:



Collaboration diagram for caffe::ReLULayer< Dtype >:



Public Member Functions

- ReLULayer (const LayerParameter &param)
- virtual const char ∗ type () const

  Returns the layer type.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Computes the error gradient w.r.t. the ReLU inputs.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Additional Inherited Members

5.77.1 Detailed Description

template<typename Dtype>
class caffe::ReLULayer< Dtype >

Rectified Linear Unit non-linearity $y = \max(0, x)$. The simple max is fast to compute, and the function does not saturate.

5.77.2 Constructor & Destructor Documentation

5.77.2.1 ReLULayer()

```
template<typename Dtype >
caffe::ReLULayer< Dtype >::ReLULayer (
            const LayerParameter & param )  [inline], [explicit]
```

Parameters

param	provides ReLUParameter relu_param, with ReLULayer options: • negative_slope (**optional**, default 0). the value ν by which negative values are multiplied.

5.77.3 Member Function Documentation

5.77.3.1 Backward_cpu()

```
template<typename Dtype >
void caffe::ReLULayer< Dtype >::Backward_cpu (
            const vector< Blob< Dtype > *> & top,
            const vector< bool > & propagate_down,
            const vector< Blob< Dtype > *> & bottom ) [protected], [virtual]
```

Computes the error gradient w.r.t. the ReLU inputs.

Parameters

top	output Blob vector (length 1), providing the error gradient with respect to the outputs 1. $(N \times C \times H \times W)$ containing error gradients $\frac{\partial E}{\partial y}$ with respect to computed outputs y
propagate_down	see Layer::Backward.
bottom	input Blob vector (length 1) 1. $(N \times C \times H \times W)$ the inputs x; Backward fills their diff with gradients $\frac{\partial E}{\partial x} = \left\{ \begin{array}{lr} 0 & \mathrm{if} \; x \le 0 \\ \frac{\partial E}{\partial y} & \mathrm{if} \; x > 0 \end{array} \right.$ if propagate_down[0], by default. If a non-zero negative_slope ν is provided, the computed gradients are $\frac{\partial E}{\partial x} = \left\{ \begin{array}{lr} \nu \frac{\partial E}{\partial y} & \mathrm{if} \; x \le 0 \\ \frac{\partial E}{\partial y} & \mathrm{if} \; x > 0 \end{array} \right.$.

Implements caffe::Layer< Dtype >.

5.77.3.2 Forward_cpu()

```
template<typename Dtype >
void caffe::ReLULayer< Dtype >::Forward_cpu (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [protected], [virtual]
```

Parameters

bottom	input Blob vector (length 1) 1. $(N \times C \times H \times W)$ the inputs x
top	output Blob vector (length 1) 1. $(N \times C \times H \times W)$ the computed outputs $y = \max(0, x)$ by default. If a non-zero negative_slope ν is provided, the computed outputs are $y = \max(0, x) + \nu \min(0, x)$.

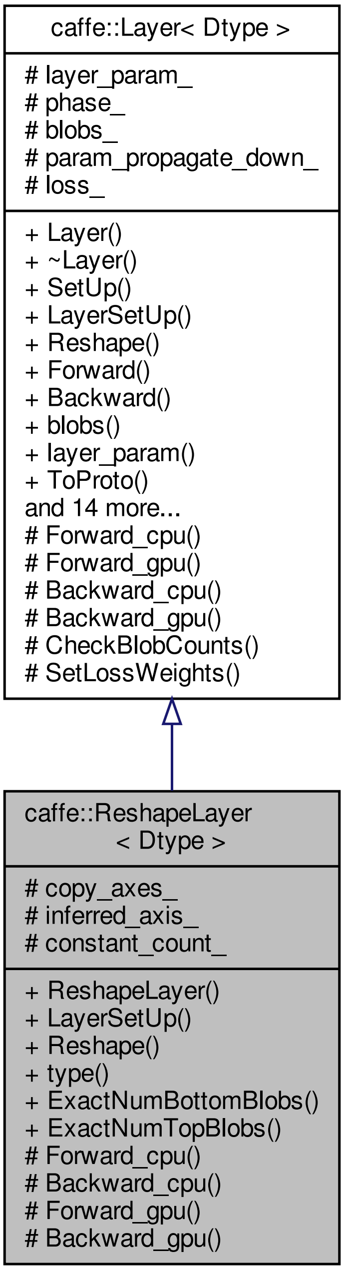
Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

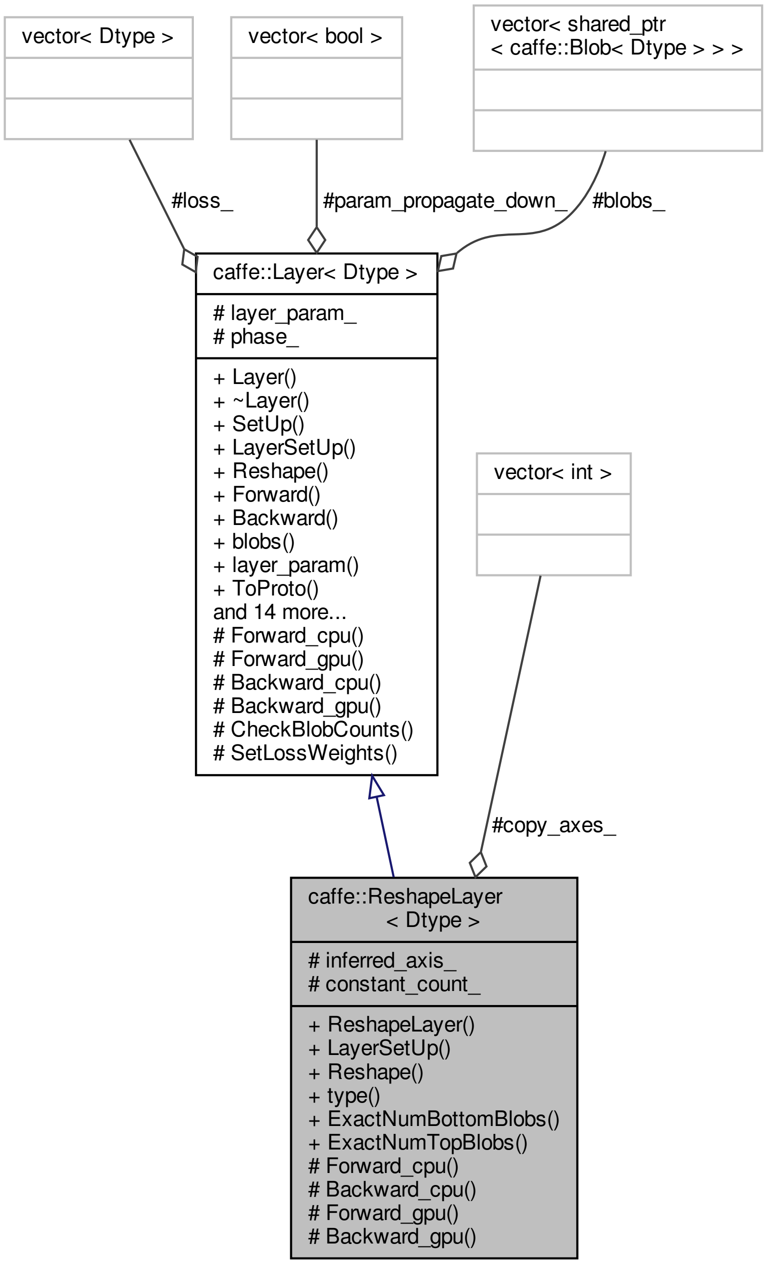
- include/caffe/layers/relu_layer.hpp
- src/caffe/layers/relu_layer.cpp

5.78 caffe::ReshapeLayer< Dtype > Class Template Reference

Inheritance diagram for caffe::ReshapeLayer< Dtype >:



Collaboration diagram for caffe::ReshapeLayer< Dtype >:



Public Member Functions

- **ReshapeLayer** (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void Reshape (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.
- virtual const char ∗ type () const

Returns the layer type.
- virtual int ExactNumBottomBlobs () const
  Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.
- virtual int ExactNumTopBlobs () const
  Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)
  Using the CPU device, compute the layer output.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)
  Using the CPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true.
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)
  Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)
  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Protected Attributes

- vector< int > copy_axes_
  vector of axes indices whose dimensions we'll copy from the bottom
- int inferred_axis_
  the index of the axis whose dimension we infer, or -1 if none
- int constant_count_
  the product of the "constant" output dimensions

5.78.1 Member Function Documentation

5.78.1.1 ExactNumBottomBlobs()

```
template<typename Dtype >
virtual int caffe::ReshapeLayer< Dtype >::ExactNumBottomBlobs ( ) const [inline], [virtual]
```

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.78.1.2 ExactNumTopBlobs()

```
template<typename Dtype >
virtual int caffe::ReshapeLayer< Dtype >::ExactNumTopBlobs ( ) const [inline], [virtual]
```

Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.78.1.3 LayerSetUp()

```
template<typename Dtype >
void caffe::ReshapeLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

5.78.1.4 Reshape()

```
template<typename Dtype >
void caffe::ReshapeLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as

reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

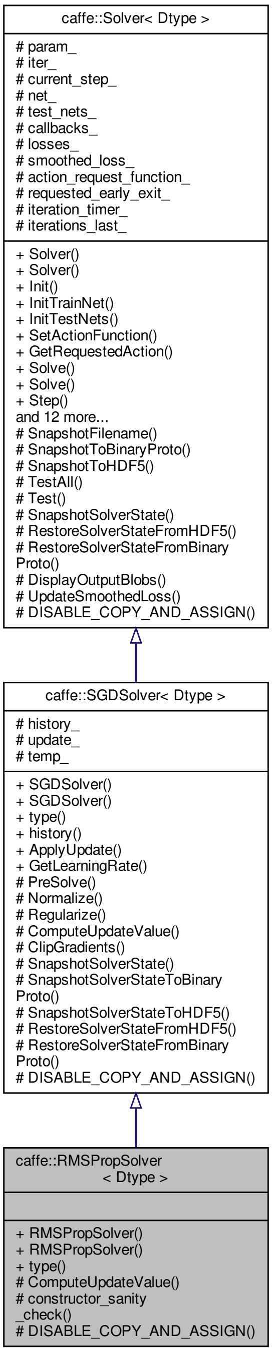
Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

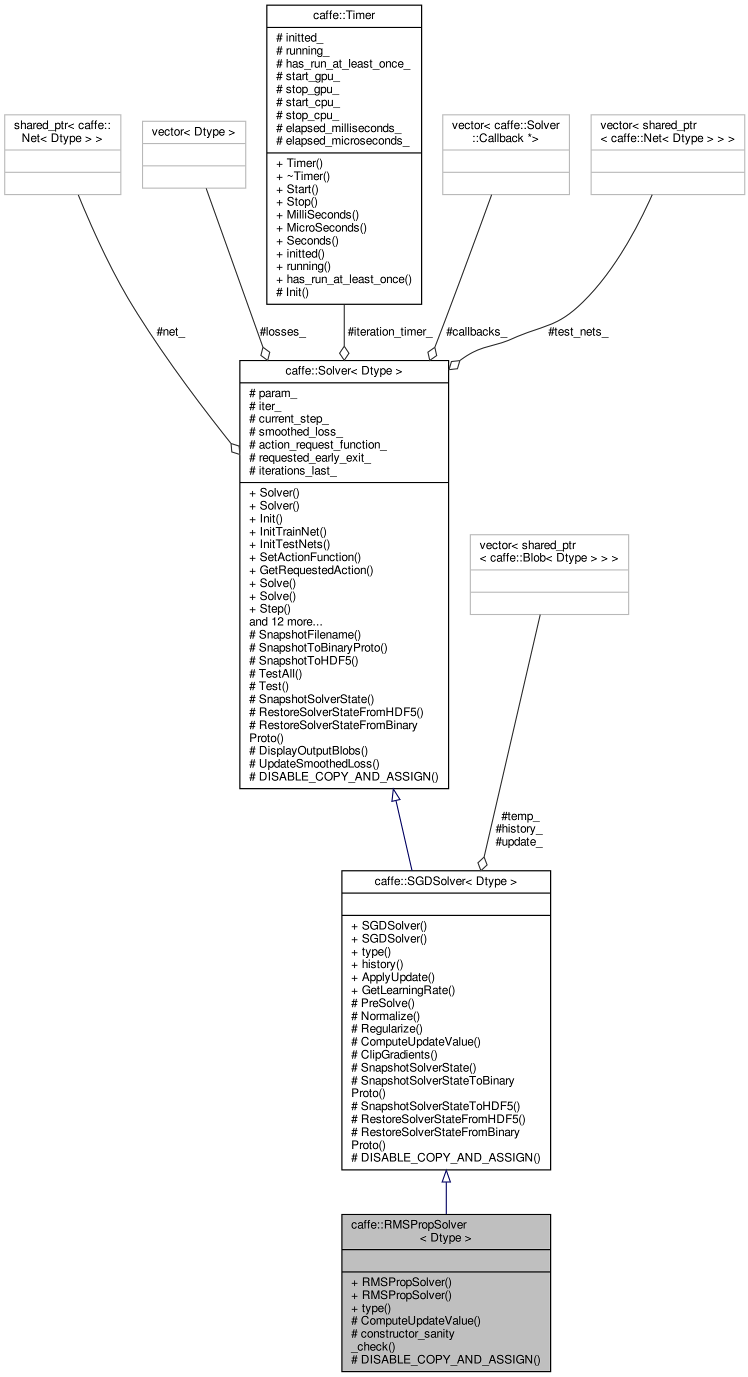
- include/caffe/layers/reshape_layer.hpp
- src/caffe/layers/reshape_layer.cpp

5.79 caffe::RMSPropSolver< Dtype > Class Template Reference

Inheritance diagram for caffe::RMSPropSolver< Dtype >:



Collaboration diagram for caffe::RMSPropSolver< Dtype >:



Public Member Functions

- **RMSPropSolver** (const SolverParameter &param)
- **RMSPropSolver** (const string &param_file)
- virtual const char ∗ type () const

  Returns the solver type.

Protected Member Functions

- virtual void **ComputeUpdateValue** (int param_id, Dtype rate)
- void **constructor_sanity_check** ()
- **DISABLE_COPY_AND_ASSIGN** (RMSPropSolver)

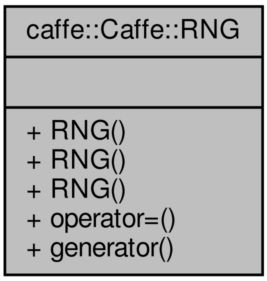
Additional Inherited Members

The documentation for this class was generated from the following files:

- include/caffe/sgd_solvers.hpp
- src/caffe/solvers/rmsprop_solver.cpp

5.80 caffe::Caffe::RNG Class Reference

Collaboration diagram for caffe::Caffe::RNG:



Classes

- class Generator

Public Member Functions

- **RNG** (unsigned int seed)
- **RNG** (const RNG &)
- RNG & **operator=** (const RNG &)
- void ∗ **generator** ()

The documentation for this class was generated from the following files:

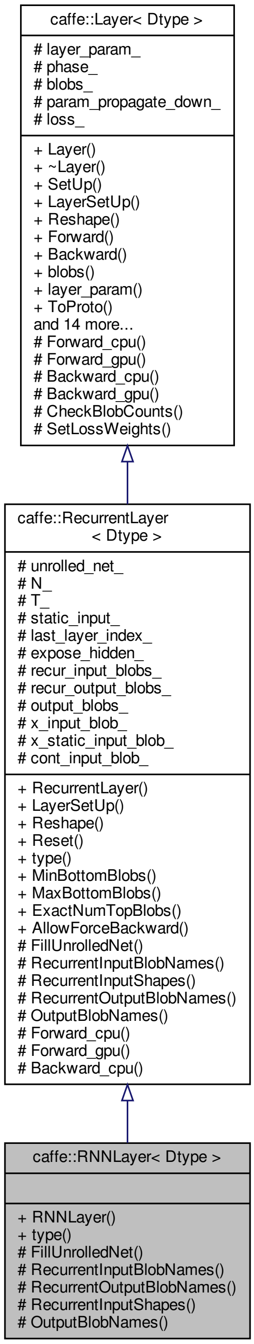
- include/caffe/common.hpp
- src/caffe/common.cpp

5.81 caffe::RNNLayer< Dtype > Class Template Reference

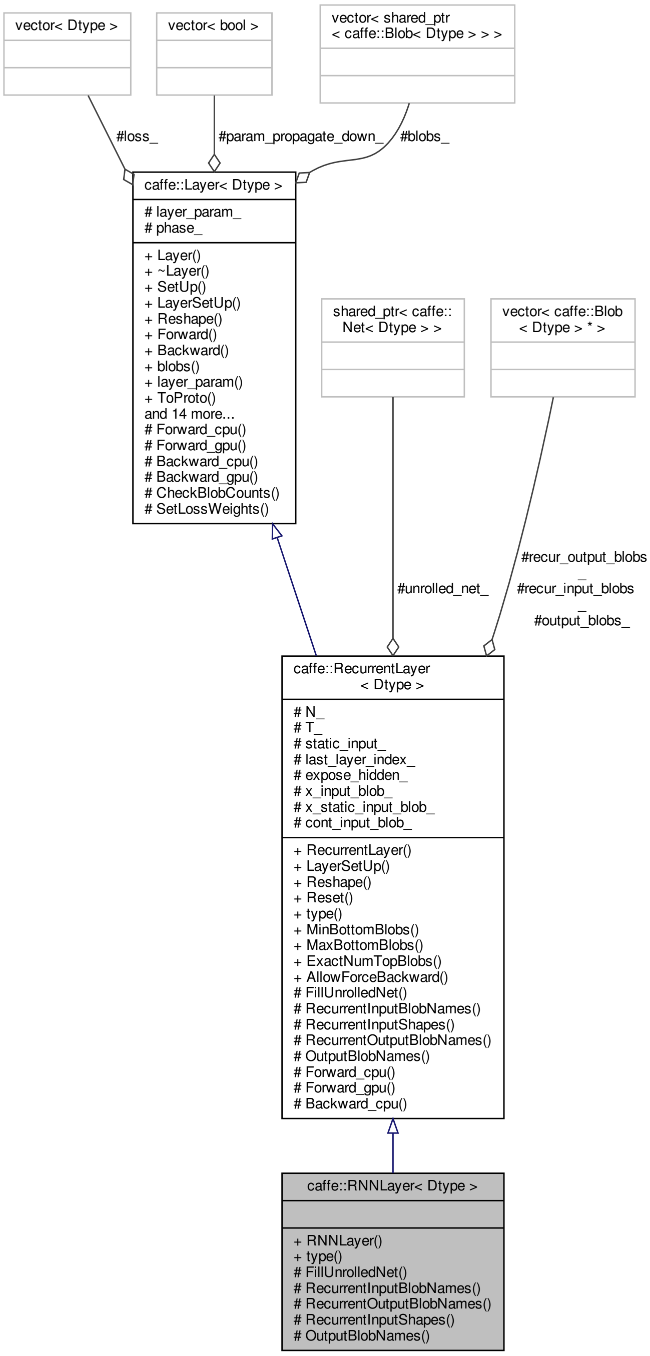
Processes time-varying inputs using a simple recurrent neural network (RNN). Implemented as a network unrolling the RNN computation in time.

```
#include <rnn_layer.hpp>
```

Inheritance diagram for caffe::RNNLayer< Dtype >:



Collaboration diagram for caffe::RNNLayer< Dtype >:



Public Member Functions

- **RNNLayer** (const LayerParameter &param)
- virtual const char ∗ type () const

  Returns the layer type.

Protected Member Functions

- virtual void FillUnrolledNet (NetParameter ∗net_param) const

  Fills net_param with the recurrent network architecture. Subclasses should define this – see RNNLayer and LSTMLayer for examples.
- virtual void RecurrentInputBlobNames (vector< string > ∗names) const

  Fills names with the names of the 0th timestep recurrent input Blob&s. Subclasses should define this – see RNNLayer and LSTMLayer for examples.
- virtual void RecurrentOutputBlobNames (vector< string > ∗names) const

  Fills names with the names of the Tth timestep recurrent output Blob&s. Subclasses should define this – see RNNLayer and LSTMLayer for examples.
- virtual void RecurrentInputShapes (vector< BlobShape > ∗shapes) const

  Fills shapes with the shapes of the recurrent input Blob&s. Subclasses should define this – see RNNLayer and LSTMLayer for examples.
- virtual void OutputBlobNames (vector< string > ∗names) const

  Fills names with the names of the output blobs, concatenated across all timesteps. Should return a name for each top Blob. Subclasses should define this – see RNNLayer and LSTMLayer for examples.

Additional Inherited Members

5.81.1 Detailed Description

template<typename Dtype>
class caffe::RNNLayer< Dtype >

Processes time-varying inputs using a simple recurrent neural network (RNN). Implemented as a network unrolling the RNN computation in time.

Given time-varying inputs x_t, computes hidden state $h_t := \tanh[W_{hh} h_{t_1} + W_{xh} x_t + b_h]$, and outputs $o_t := \tanh[W_{ho} h_t + b_o]$.

The documentation for this class was generated from the following files:

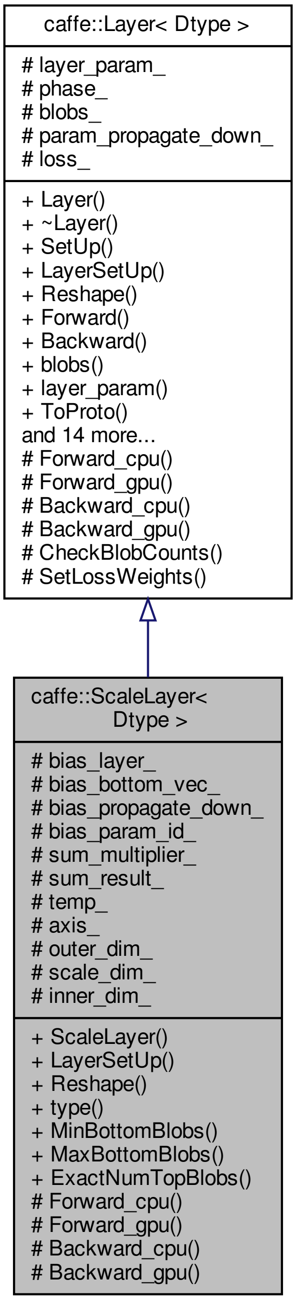
- include/caffe/layers/rnn_layer.hpp
- src/caffe/layers/rnn_layer.cpp

5.82 caffe::ScaleLayer< Dtype > Class Template Reference

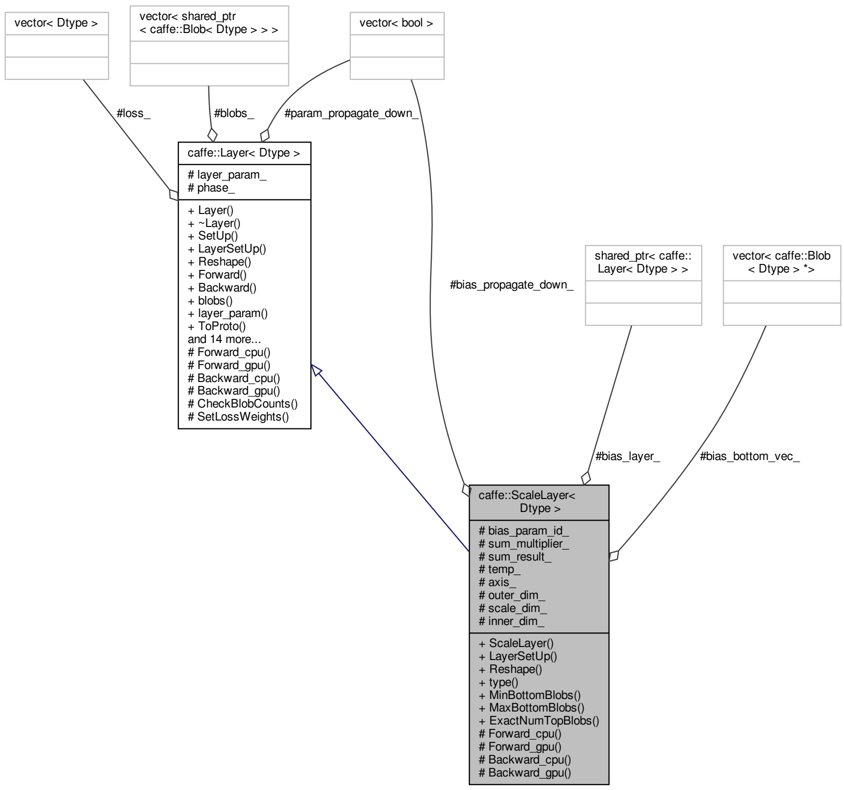
Computes the elementwise product of two input Blobs, with the shape of the latter Blob "broadcast" to match the shape of the former. Equivalent to tiling the latter Blob, then computing the elementwise product. Note: for efficiency and convenience, this layer can additionally perform a "broadcast" sum too when `bias_term: true` is set.

```
#include <scale_layer.hpp>
```

Inheritance diagram for caffe::ScaleLayer< Dtype >:



Collaboration diagram for caffe::ScaleLayer< Dtype >:



Public Member Functions

- **ScaleLayer** (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void Reshape (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.
- virtual const char * type () const

  Returns the layer type.
- virtual int MinBottomBlobs () const

  Returns the minimum number of bottom blobs required by the layer, or -1 if no minimum number is required.
- virtual int MaxBottomBlobs () const

  Returns the maximum number of bottom blobs required by the layer, or -1 if no maximum number is required.
- virtual int ExactNumTopBlobs () const

  Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.

- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the CPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true.

- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Protected Attributes

- shared_ptr< Layer< Dtype > > **bias_layer_**
- vector< Blob< Dtype > * > **bias_bottom_vec_**
- vector< bool > **bias_propagate_down_**
- int **bias_param_id_**
- Blob< Dtype > **sum_multiplier_**
- Blob< Dtype > **sum_result_**
- Blob< Dtype > **temp_**
- int **axis_**
- int **outer_dim_**
- int **scale_dim_**
- int **inner_dim_**

5.82.1 Detailed Description

template<typename Dtype>
class caffe::ScaleLayer< Dtype >

Computes the elementwise product of two input Blobs, with the shape of the latter Blob "broadcast" to match the shape of the former. Equivalent to tiling the latter Blob, then computing the elementwise product. Note: for efficiency and convenience, this layer can additionally perform a "broadcast" sum too when `bias_term: true` is set.

The latter, scale input may be omitted, in which case it's learned as parameter of the layer (as is the bias, if it is included).

5.82.2 Member Function Documentation

5.82.2.1 ExactNumTopBlobs()

```
template<typename Dtype >
virtual int caffe::ScaleLayer< Dtype >::ExactNumTopBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.82.2.2 Forward_cpu()

```
template<typename Dtype >
void caffe::ScaleLayer< Dtype >::Forward_cpu (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [protected], [virtual]
```

In the below shape specifications, i denotes the value of the `axis` field given by `this->layer_param_.scale_param().axis()`, after canonicalization (i.e., conversion from negative to positive index, if applicable).

Parameters

bottom	input Blob vector (length 2) 1. $(d_0 \times ... \times d_i \times ... \times d_j \times ... \times d_n)$ the first factor x 2. $(d_i \times ... \times d_j)$ the second factor y
top	output Blob vector (length 1) 1. $(d_0 \times ... \times d_i \times ... \times d_j \times ... \times d_n)$ the product $z = xy$ computed after "broadcasting" y. Equivalent to tiling y to have the same shape as x, then computing the elementwise product.

Implements caffe::Layer< Dtype >.

5.82.2.3 LayerSetUp()

```
template<typename Dtype >
void caffe::ScaleLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

5.82.2.4 MaxBottomBlobs()

```
template<typename Dtype >
virtual int caffe::ScaleLayer< Dtype >::MaxBottomBlobs ( ) const  [inline], [virtual]
```

Returns the maximum number of bottom blobs required by the layer, or -1 if no maximum number is required.

This method should be overridden to return a non-negative value if your layer expects some maximum number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.82.2.5 MinBottomBlobs()

```
template<typename Dtype >
virtual int caffe::ScaleLayer< Dtype >::MinBottomBlobs ( ) const  [inline], [virtual]
```

Returns the minimum number of bottom blobs required by the layer, or -1 if no minimum number is required.

This method should be overridden to return a non-negative value if your layer expects some minimum number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.82.2.6 Reshape()

```
template<typename Dtype >
void caffe::ScaleLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

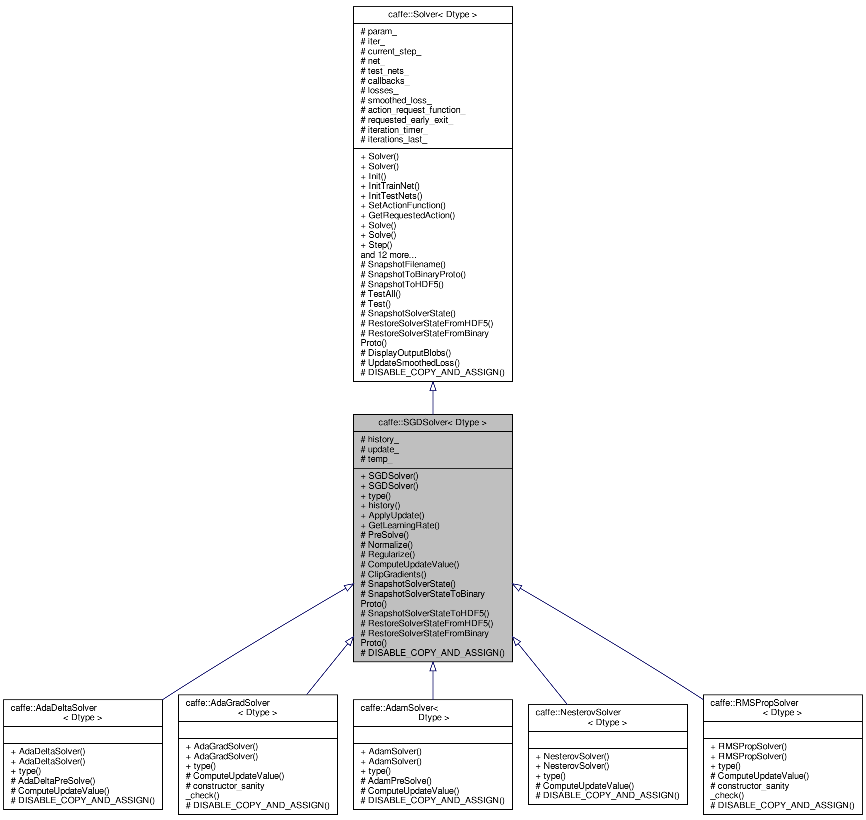
- include/caffe/layers/scale_layer.hpp
- src/caffe/layers/scale_layer.cpp

5.83 caffe::SGDSolver< Dtype > Class Template Reference

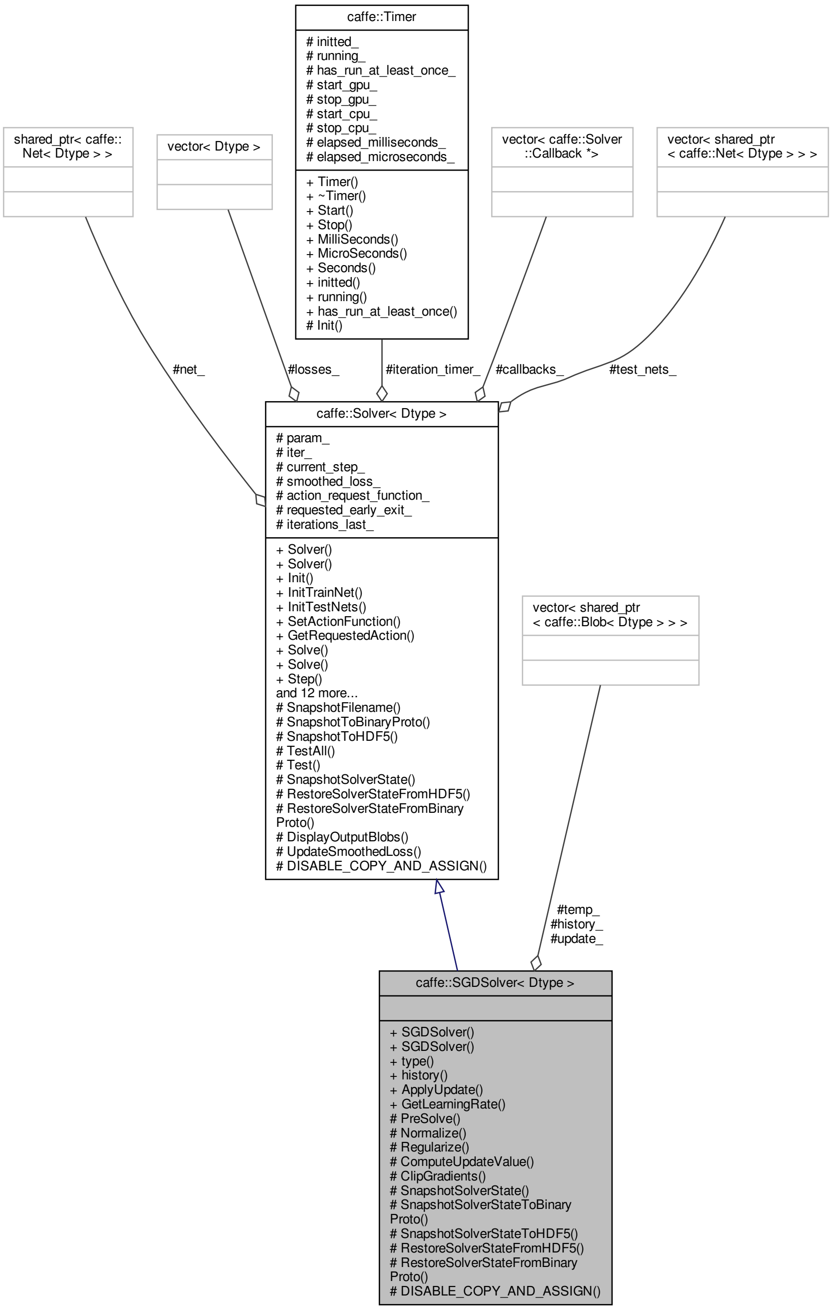
Optimizes the parameters of a Net using stochastic gradient descent (SGD) with momentum.

```
#include <sgd_solvers.hpp>
```

Inheritance diagram for caffe::SGDSolver< Dtype >:



Collaboration diagram for caffe::SGDSolver< Dtype >:



Public Member Functions

- **SGDSolver** (const SolverParameter &param)
- **SGDSolver** (const string &param_file)
- virtual const char $*$ type () const

  Returns the solver type.
- const vector< shared_ptr< Blob< Dtype > > > & **history** ()
- virtual void **ApplyUpdate** ()
- Dtype **GetLearningRate** ()

Protected Member Functions

- void **PreSolve** ()
- virtual void **Normalize** (int param_id)
- virtual void **Regularize** (int param_id)
- virtual void **ComputeUpdateValue** (int param_id, Dtype rate)
- virtual void **ClipGradients** ()
- virtual void **SnapshotSolverState** (const string &model_filename)
- virtual void **SnapshotSolverStateToBinaryProto** (const string &model_filename)
- virtual void **SnapshotSolverStateToHDF5** (const string &model_filename)
- virtual void **RestoreSolverStateFromHDF5** (const string &state_file)
- virtual void **RestoreSolverStateFromBinaryProto** (const string &state_file)
- **DISABLE_COPY_AND_ASSIGN** (SGDSolver)

Protected Attributes

- vector< shared_ptr< Blob< Dtype > > > **history_**
- vector< shared_ptr< Blob< Dtype > > > **update_**
- vector< shared_ptr< Blob< Dtype > > > **temp_**

5.83.1 Detailed Description

template<typename Dtype>
class caffe::SGDSolver< Dtype >

Optimizes the parameters of a Net using stochastic gradient descent (SGD) with momentum.

The documentation for this class was generated from the following files:

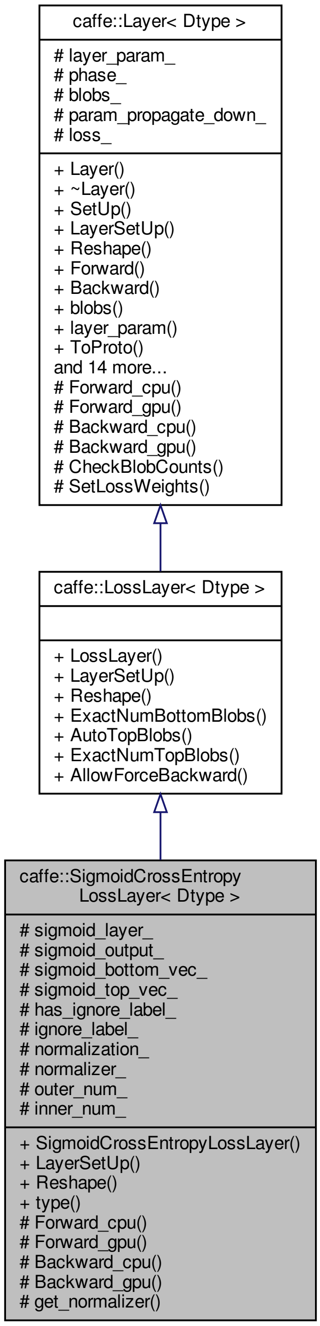
- include/caffe/sgd_solvers.hpp
- src/caffe/solvers/sgd_solver.cpp

5.84 caffe::SigmoidCrossEntropyLossLayer< Dtype > Class Template Reference

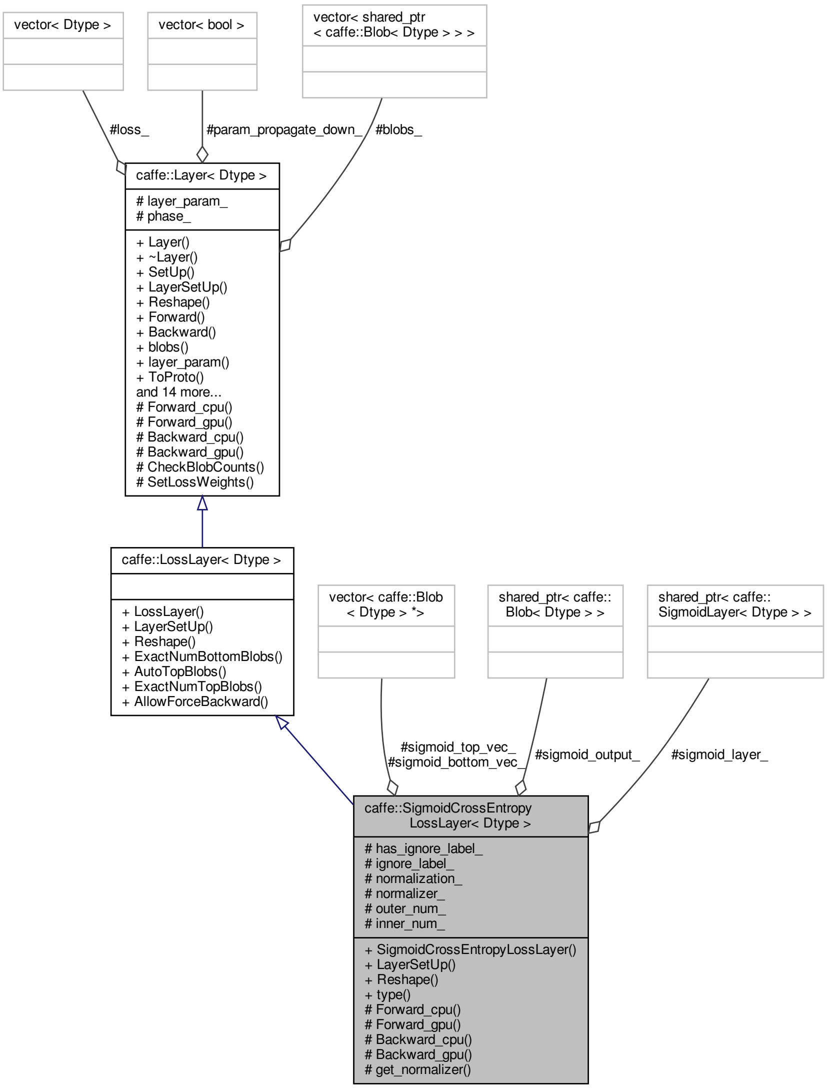
Computes the cross-entropy (logistic) loss $E = \frac{-1}{n} \sum\limits_{n=1}^{N} \left[p_n \log \hat{p}_n + (1 - p_n) \log(1 - \hat{p}_n) \right]$, often used for predicting targets interpreted as probabilities.

`#include <sigmoid_cross_entropy_loss_layer.hpp>`

Inheritance diagram for caffe::SigmoidCrossEntropyLossLayer< Dtype >:



Collaboration diagram for caffe::SigmoidCrossEntropyLossLayer< Dtype >:



Public Member Functions

- **SigmoidCrossEntropyLossLayer** (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void Reshape (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.
- virtual const char ∗ type () const

  Returns the layer type.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

Computes the cross-entropy (logistic) loss $E = \frac{-1}{n} \sum\limits_{n=1}^{N} \left[p_n \log \hat{p}_n + (1 - p_n) \log(1 - \hat{p}_n) \right]$, often used for predicting targets interpreted as probabilities.

- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Computes the sigmoid cross-entropy loss error gradient w.r.t. the predictions.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.
- virtual Dtype get_normalizer (LossParameter_NormalizationMode normalization_mode, int valid_count)

Protected Attributes

- shared_ptr< SigmoidLayer< Dtype > > sigmoid_layer_

  The internal SigmoidLayer used to map predictions to probabilities.
- shared_ptr< Blob< Dtype > > sigmoid_output_

  sigmoid_output stores the output of the SigmoidLayer.
- vector< Blob< Dtype > * > sigmoid_bottom_vec_

  bottom vector holder to call the underlying SigmoidLayer::Forward
- vector< Blob< Dtype > * > sigmoid_top_vec_

  top vector holder to call the underlying SigmoidLayer::Forward
- bool has_ignore_label_

  Whether to ignore instances with a certain label.
- int ignore_label_

  The label indicating that an instance should be ignored.
- LossParameter_NormalizationMode normalization_

  How to normalize the loss.
- Dtype **normalizer_**
- int **outer_num_**
- int **inner_num_**

5.84.1 Detailed Description

template<typename Dtype>
class caffe::SigmoidCrossEntropyLossLayer< Dtype >

Computes the cross-entropy (logistic) loss $E = \frac{-1}{n} \sum\limits_{n=1}^{N} \left[p_n \log \hat{p}_n + (1 - p_n) \log(1 - \hat{p}_n) \right]$, often used for predicting targets interpreted as probabilities.

This layer is implemented rather than separate SigmoidLayer + CrossEntropyLayer as its gradient computation is more numerically stable. At test time, this layer can be replaced simply by a SigmoidLayer.

Parameters

bottom	input Blob vector (length 2) 1. $(N \times C \times H \times W)$ the scores $x \in [-\infty, +\infty]$, which this layer maps to probability predictions $\hat{p}_n = \sigma(x_n) \in [0, 1]$ using the sigmoid function $\sigma(.)$ (see SigmoidLayer). 2. $(N \times C \times H \times W)$ the targets $y \in [0, 1]$
top	output Blob vector (length 1) 1. $(1 \times 1 \times 1 \times 1)$ the computed cross-entropy loss: $E = \frac{-1}{n} \sum\limits_{n=1}^{N} \left[p_n \log \hat{p}_n + (1 - p_n) \log(1 - \hat{p}_n) \right]$

5.84.2 Member Function Documentation

5.84.2.1 Backward_cpu()

```
template<typename Dtype >
void caffe::SigmoidCrossEntropyLossLayer< Dtype >::Backward_cpu (
            const vector< Blob< Dtype > *> & top,
            const vector< bool > & propagate_down,
            const vector< Blob< Dtype > *> & bottom )  [protected], [virtual]
```

Computes the sigmoid cross-entropy loss error gradient w.r.t. the predictions.

Gradients cannot be computed with respect to the target inputs (bottom[1]), so this method ignores bottom[1] and requires !propagate_down[1], crashing if propagate_down[1] is set.

Parameters

top	output Blob vector (length 1), providing the error gradient with respect to the outputs 1. $(1 \times 1 \times 1 \times 1)$ This Blob's diff will simply contain the loss_weight$*$ λ, as λ is the coefficient of this layer's output ℓ_i in the overall Net loss $E = \lambda_i \ell_i + \mbox{other loss terms}$; hence $\frac{\partial E}{\partial \ell_i} = \lambda_i$. ($*$Assuming that this top Blob is not used as a bottom (input) by any other layer of the Net.)
propagate_down	see Layer::Backward. propagate_down[1] must be false as gradient computation with respect to the targets is not implemented.
bottom	input Blob vector (length 2) 1. $(N \times C \times H \times W)$ the predictions x; Backward computes diff $\frac{\partial E}{\partial x} = \frac{1}{n} \sum\limits_{n=1}^{N} (\hat{p}_n - p_n)$ 2. $(N \times 1 \times 1 \times 1)$ the labels – ignored as we can't compute their error gradients

Implements caffe::Layer< Dtype >.

5.84.2.2 Forward_cpu()

```
template<typename Dtype >
void caffe::SigmoidCrossEntropyLossLayer< Dtype >::Forward_cpu (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [protected], [virtual]
```

Computes the cross-entropy (logistic) loss $E = \frac{-1}{n} \sum\limits_{n=1}^N \left[p_n \log \hat{p}_n + (1 - p_n) \log(1 - \hat{p}_n) \right]$, often used for predicting targets interpreted as probabilities.

This layer is implemented rather than separate SigmoidLayer + CrossEntropyLayer as its gradient computation is more numerically stable. At test time, this layer can be replaced simply by a SigmoidLayer.

Parameters

bottom	input Blob vector (length 2) 1. $(N \times C \times H \times W)$ the scores $x \in [-\infty, +\infty]$, which this layer maps to probability predictions $\hat{p}_n = \sigma(x_n) \in [0, 1]$ using the sigmoid function $\sigma(.)$ (see SigmoidLayer). 2. $(N \times C \times H \times W)$ the targets $y \in [0, 1]$
top	output Blob vector (length 1) 1. $(1 \times 1 \times 1 \times 1)$ the computed cross-entropy loss: $E = \frac{-1}{n} \sum\limits_{n=1}^N \left[p_n \log \hat{p}_n + (1 - p_n) \log(1 - \hat{p}_n) \right]$

Implements caffe::Layer< Dtype >.

5.84.2.3 get_normalizer()

```
template<typename Dtype >
Dtype caffe::SigmoidCrossEntropyLossLayer< Dtype >::get_normalizer (
            LossParameter_NormalizationMode normalization_mode,
            int valid_count ) [protected], [virtual]
```

Read the normalization mode parameter and compute the normalizer based on the blob size. If normalization_↩ mode is VALID, the count of valid outputs will be read from valid_count, unless it is -1 in which case all outputs are assumed to be valid.

5.84.2.4 LayerSetUp()

```
template<typename Dtype >
void caffe::SigmoidCrossEntropyLossLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::LossLayer< Dtype >.

5.84.2.5 Reshape()

```
template<typename Dtype >
void caffe::SigmoidCrossEntropyLossLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Reimplemented from caffe::LossLayer< Dtype >.

The documentation for this class was generated from the following files:

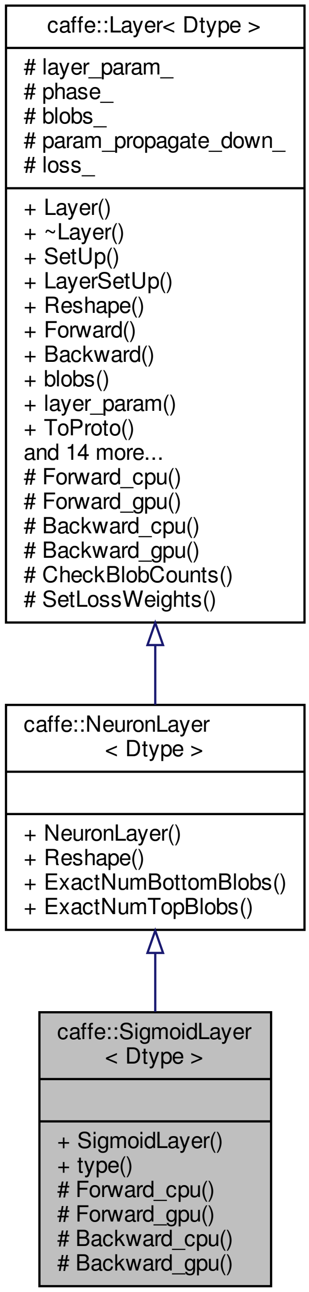
- include/caffe/layers/sigmoid_cross_entropy_loss_layer.hpp
- src/caffe/layers/sigmoid_cross_entropy_loss_layer.cpp

5.85 caffe::SigmoidLayer< Dtype > Class Template Reference

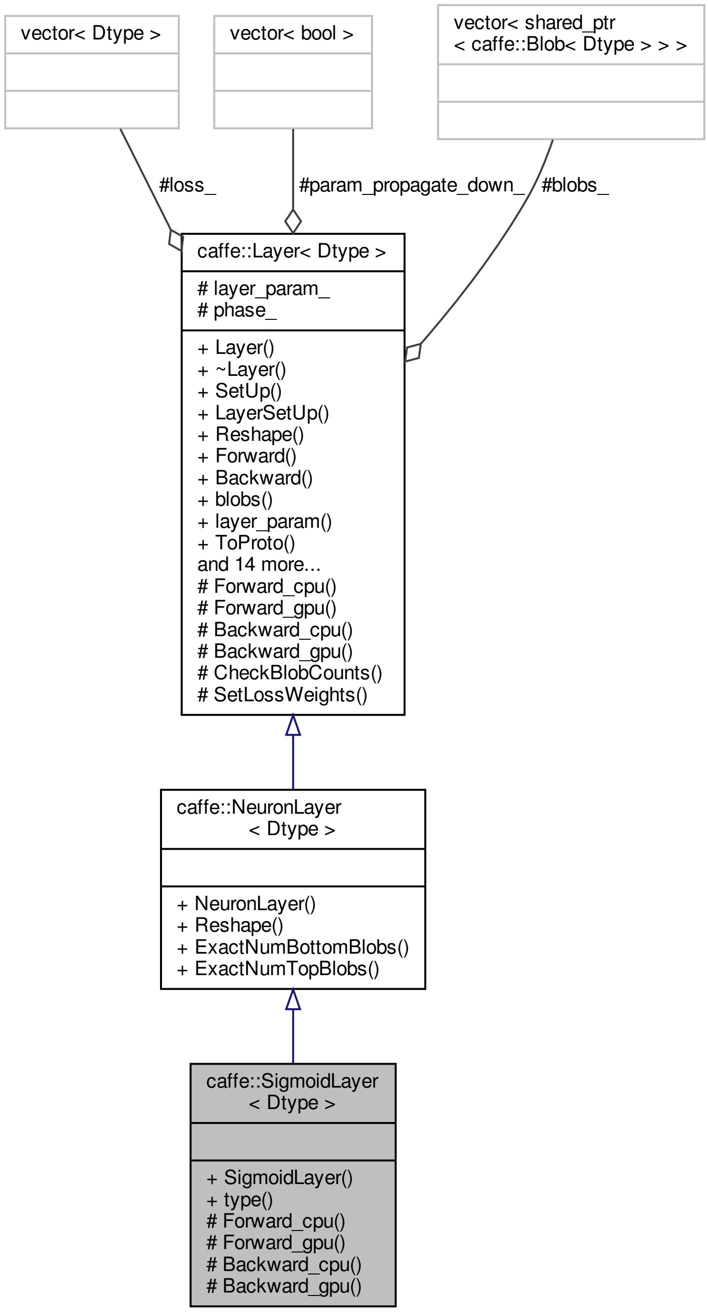
Sigmoid function non-linearity $y = (1 + \exp(-x))^{-1}$, a classic choice in neural networks.

`#include <sigmoid_layer.hpp>`

Inheritance diagram for caffe::SigmoidLayer< Dtype >:



Collaboration diagram for caffe::SigmoidLayer< Dtype >:



Public Member Functions

- **SigmoidLayer** (const LayerParameter &param)
- virtual const char * type () const

  Returns the layer type.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

    Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

    Computes the error gradient w.r.t. the sigmoid inputs.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

    Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Additional Inherited Members

5.85.1 Detailed Description

template<typename Dtype>
class caffe::SigmoidLayer< Dtype >

Sigmoid function non-linearity $y = (1 + \exp(-x))^{-1}$, a classic choice in neural networks.

Note that the gradient vanishes as the values move away from 0. The ReLULayer is often a better choice for this reason.

5.85.2 Member Function Documentation

5.85.2.1 Backward_cpu()

```
template<typename Dtype >
void caffe::SigmoidLayer< Dtype >::Backward_cpu (
            const vector< Blob< Dtype > *> & top,
            const vector< bool > & propagate_down,
            const vector< Blob< Dtype > *> & bottom )  [protected], [virtual]
```

Computes the error gradient w.r.t. the sigmoid inputs.

Parameters

top	output Blob vector (length 1), providing the error gradient with respect to the outputs 1. $(N \times C \times H \times W)$ containing error gradients $\frac{\partial E}{\partial y}$ with respect to computed outputs y
propagate_down	see Layer::Backward.
bottom	input Blob vector (length 1) 1. $(N \times C \times H \times W)$ the inputs x; Backward fills their diff with gradients $\frac{\partial E}{\partial x} = \frac{\partial E}{\partial y} y(1 - y)$ if propagate_down[0]

Implements caffe::Layer< Dtype >.

5.85.2.2 Forward_cpu()

```
template<typename Dtype >
void caffe::SigmoidLayer< Dtype >::Forward_cpu (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [protected], [virtual]
```

Parameters

bottom	input Blob vector (length 1) 1. $(N \times C \times H \times W)$ the inputs x
top	output Blob vector (length 1) 1. $(N \times C \times H \times W)$ the computed outputs $y = (1 + \exp(-x))^{-1}$

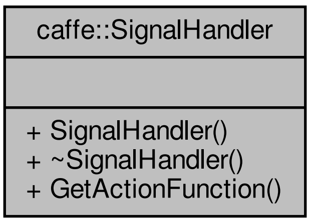
Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

- include/caffe/layers/sigmoid_layer.hpp
- src/caffe/layers/sigmoid_layer.cpp

5.86 caffe::SignalHandler Class Reference

Collaboration diagram for caffe::SignalHandler:



Public Member Functions

- **SignalHandler** (SolverAction::Enum SIGINT_action, SolverAction::Enum SIGHUP_action)
- ActionCallback **GetActionFunction** ()

The documentation for this class was generated from the following files:

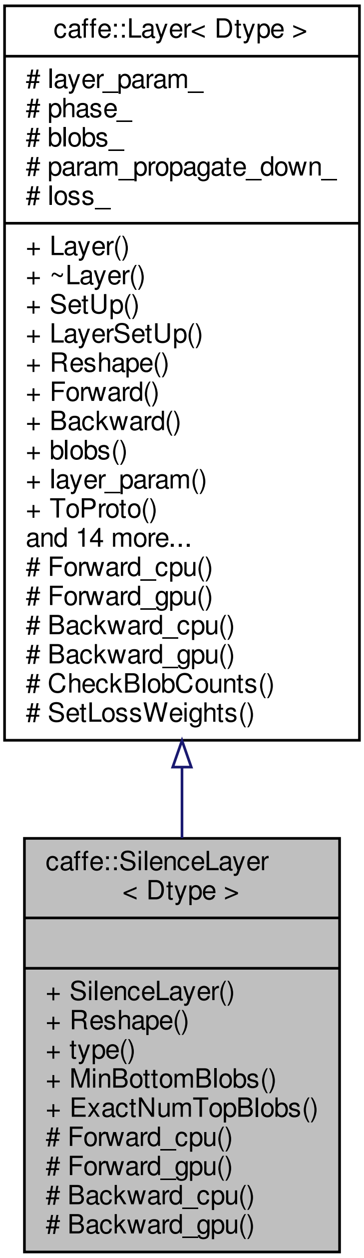
- include/caffe/util/signal_handler.h
- src/caffe/util/signal_handler.cpp

5.87 caffe::SilenceLayer< Dtype > Class Template Reference

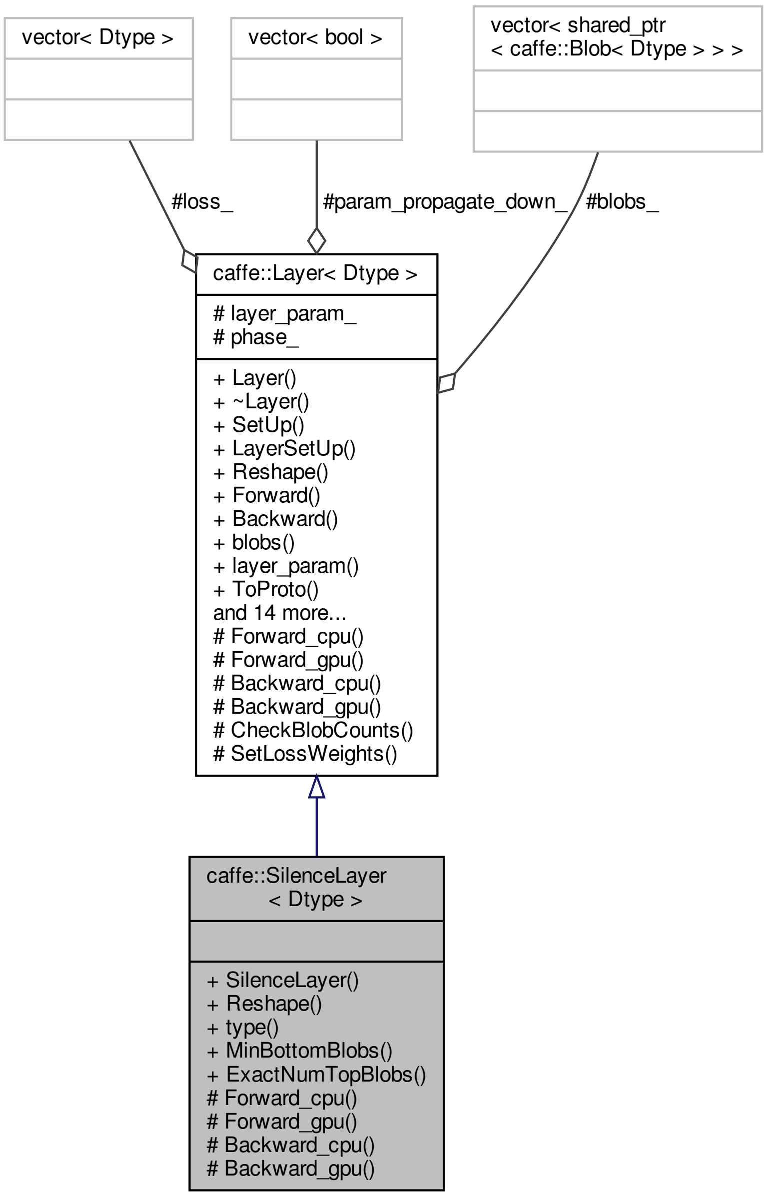
Ignores bottom blobs while producing no top blobs. (This is useful to suppress outputs during testing.)

```
#include <silence_layer.hpp>
```

Inheritance diagram for caffe::SilenceLayer< Dtype >:



Collaboration diagram for caffe::SilenceLayer< Dtype >:



Public Member Functions

- **SilenceLayer** (const LayerParameter &param)
- virtual void Reshape (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.
- virtual const char ∗ type () const

  Returns the layer type.
- virtual int MinBottomBlobs () const

  Returns the minimum number of bottom blobs required by the layer, or -1 if no minimum number is required.
- virtual int ExactNumTopBlobs () const

  Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the CPU device, compute the layer output.
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the CPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Additional Inherited Members

5.87.1 Detailed Description

template<typename Dtype>
class caffe::SilenceLayer< Dtype >

Ignores bottom blobs while producing no top blobs. (This is useful to suppress outputs during testing.)

5.87.2 Member Function Documentation

5.87.2.1 ExactNumTopBlobs()

```
template<typename Dtype >
virtual int caffe::SilenceLayer< Dtype >::ExactNumTopBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.87.2.2 MinBottomBlobs()

```
template<typename Dtype >
virtual int caffe::SilenceLayer< Dtype >::MinBottomBlobs ( ) const  [inline], [virtual]
```

Returns the minimum number of bottom blobs required by the layer, or -1 if no minimum number is required.

This method should be overridden to return a non-negative value if your layer expects some minimum number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.87.2.3 Reshape()

```
template<typename Dtype >
virtual void caffe::SilenceLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [inline], [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

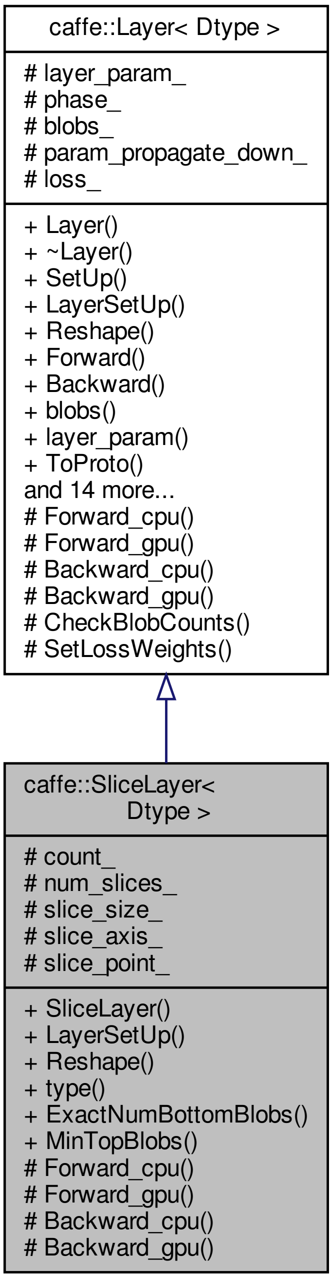
- include/caffe/layers/silence_layer.hpp
- src/caffe/layers/silence_layer.cpp

5.88 caffe::SliceLayer< Dtype > Class Template Reference

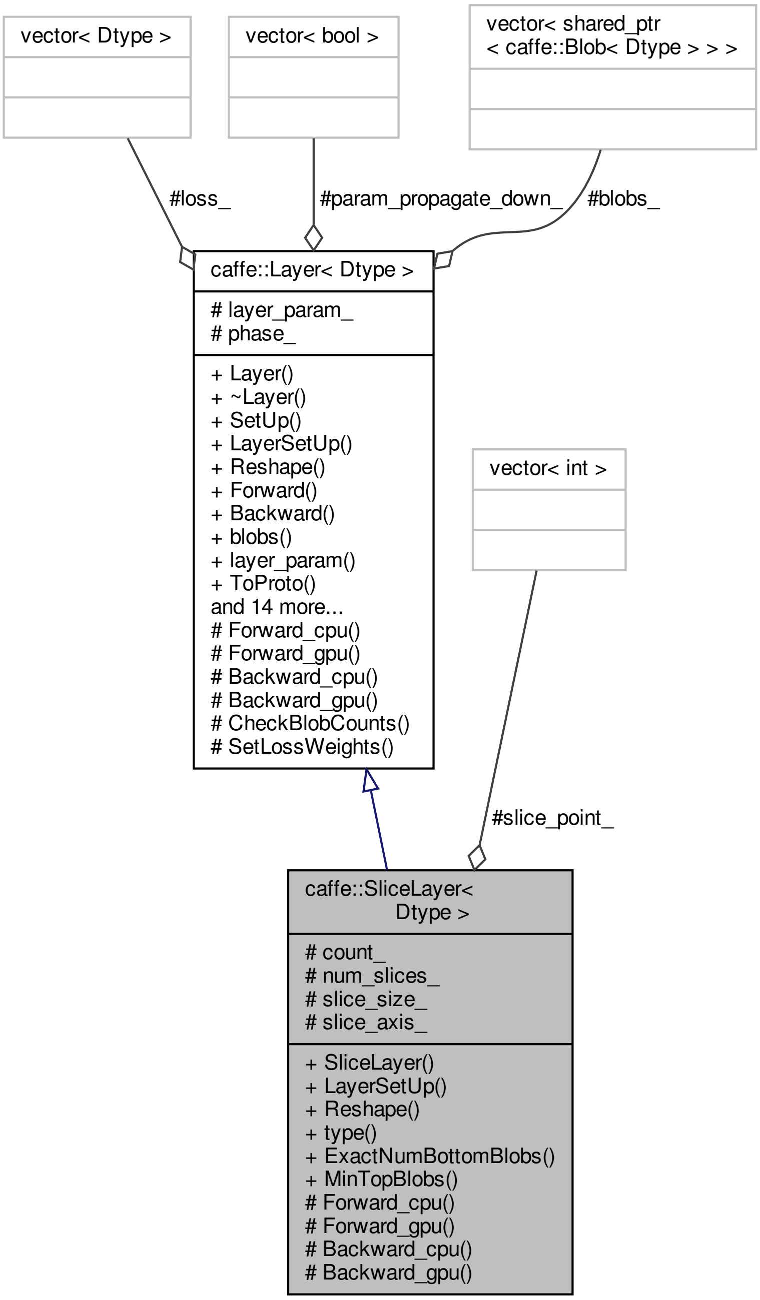
Takes a Blob and slices it along either the num or channel dimension, outputting multiple sliced Blob results.

```
#include <slice_layer.hpp>
```

Inheritance diagram for caffe::SliceLayer< Dtype >:



Collaboration diagram for caffe::SliceLayer< Dtype >:



Public Member Functions

- **SliceLayer** (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void Reshape (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

- virtual const char * type () const

  Returns the layer type.
- virtual int ExactNumBottomBlobs () const

  Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.
- virtual int MinTopBlobs () const

  Returns the minimum number of top blobs required by the layer, or -1 if no minimum number is required.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the CPU device, compute the layer output.
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the CPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Protected Attributes

- int **count_**
- int **num_slices_**
- int **slice_size_**
- int **slice_axis_**
- vector< int > **slice_point_**

5.88.1 Detailed Description

template<typename Dtype>
class caffe::SliceLayer< Dtype >

Takes a Blob and slices it along either the num or channel dimension, outputting multiple sliced Blob results.

TODO(dox): thorough documentation for Forward, Backward, and proto params.

5.88.2 Member Function Documentation

5.88.2.1 ExactNumBottomBlobs()

```
template<typename Dtype >
virtual int caffe::SliceLayer< Dtype >::ExactNumBottomBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.88.2.2 LayerSetUp()

```
template<typename Dtype >
void caffe::SliceLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

5.88.2.3 MinTopBlobs()

```
template<typename Dtype >
virtual int caffe::SliceLayer< Dtype >::MinTopBlobs ( ) const  [inline], [virtual]
```

Returns the minimum number of top blobs required by the layer, or -1 if no minimum number is required.

This method should be overridden to return a non-negative value if your layer expects some minimum number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.88.2.4 Reshape()

```
template<typename Dtype >
void caffe::SliceLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

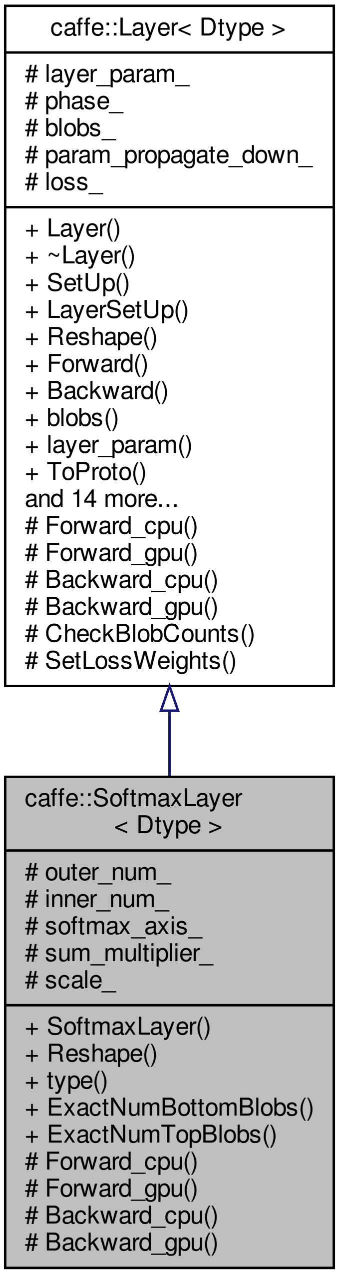
- include/caffe/layers/slice_layer.hpp
- src/caffe/layers/slice_layer.cpp

5.89 caffe::SoftmaxLayer< Dtype > Class Template Reference

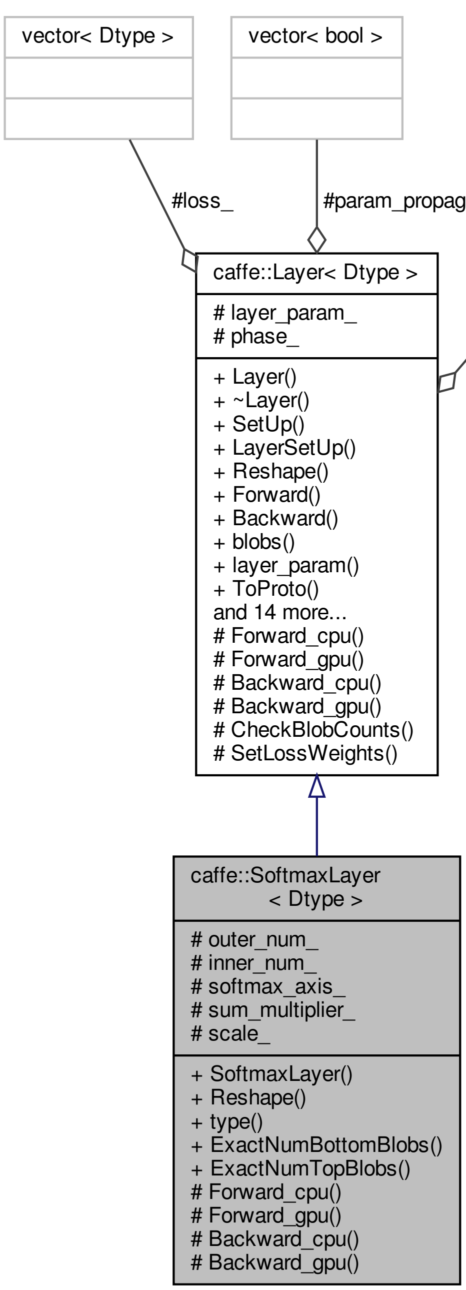
Computes the softmax function.

`#include <softmax_layer.hpp>`

Inheritance diagram for caffe::SoftmaxLayer< Dtype >:



Collaboration diagram for caffe::SoftmaxLayer< Dtype >:



Public Member Functions

- **SoftmaxLayer** (const LayerParameter &param)
- virtual void Reshape (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

    Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.
- virtual const char ∗ type () const

    Returns the layer type.

- virtual int ExactNumBottomBlobs () const

    Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.
- virtual int ExactNumTopBlobs () const

    Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

    Using the CPU device, compute the layer output.
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

    Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

    Using the CPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

    Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Protected Attributes

- int **outer_num_**
- int **inner_num_**
- int **softmax_axis_**
- Blob< Dtype > sum_multiplier_

    sum_multiplier is used to carry out sum using BLAS
- Blob< Dtype > scale_

    scale is an intermediate Blob to hold temporary results.

5.89.1 Detailed Description

template<typename Dtype>
class caffe::SoftmaxLayer< Dtype >

Computes the softmax function.

TODO(dox): thorough documentation for Forward, Backward, and proto params.

5.89.2 Member Function Documentation

5.89.2.1 ExactNumBottomBlobs()

```
template<typename Dtype >
virtual int caffe::SoftmaxLayer< Dtype >::ExactNumBottomBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.89.2.2 ExactNumTopBlobs()

```
template<typename Dtype >
virtual int caffe::SoftmaxLayer< Dtype >::ExactNumTopBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.89.2.3 Reshape()

```
template<typename Dtype >
void caffe::SoftmaxLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

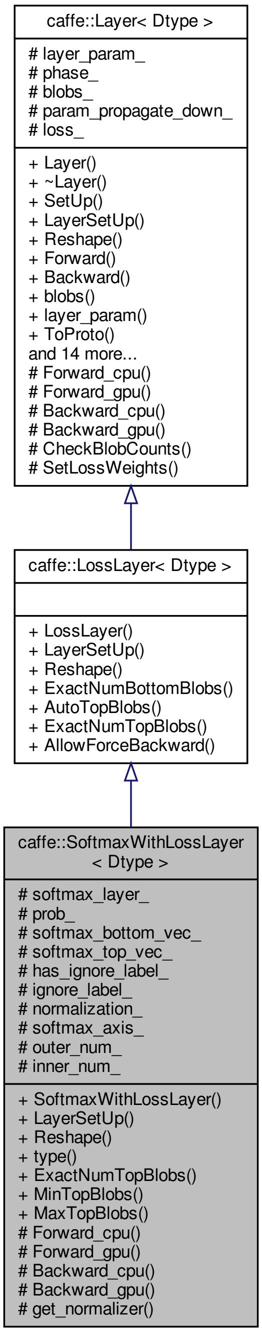
- include/caffe/layers/softmax_layer.hpp
- src/caffe/layers/softmax_layer.cpp

5.90 caffe::SoftmaxWithLossLayer< Dtype > Class Template Reference

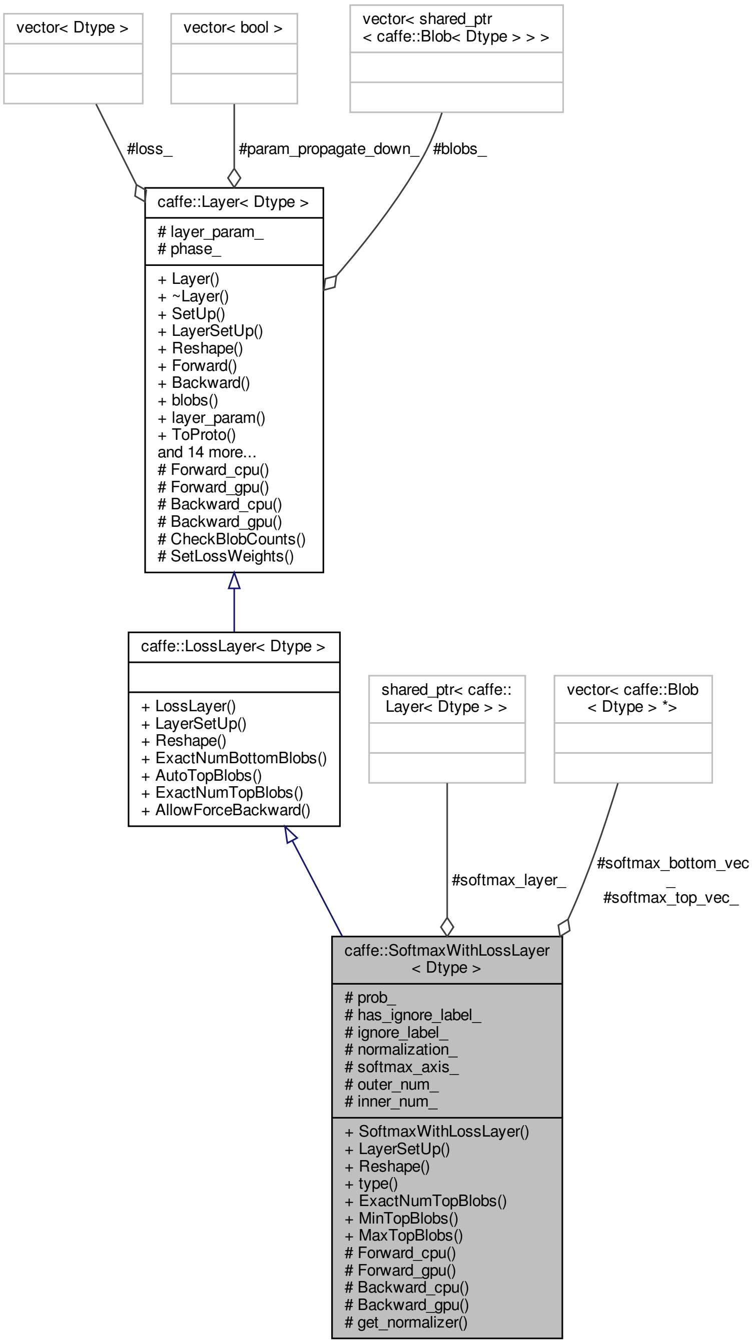
Computes the multinomial logistic loss for a one-of-many classification task, passing real-valued predictions through a softmax to get a probability distribution over classes.

`#include <softmax_loss_layer.hpp>`

Inheritance diagram for caffe::SoftmaxWithLossLayer< Dtype >:



Collaboration diagram for caffe::SoftmaxWithLossLayer< Dtype >:



Public Member Functions

- SoftmaxWithLossLayer (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void Reshape (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

- virtual const char * type () const

  Returns the layer type.
- virtual int ExactNumTopBlobs () const

  Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.
- virtual int MinTopBlobs () const

  Returns the minimum number of top blobs required by the layer, or -1 if no minimum number is required.
- virtual int MaxTopBlobs () const

  Returns the maximum number of top blobs required by the layer, or -1 if no maximum number is required.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the CPU device, compute the layer output.
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Computes the softmax loss error gradient w.r.t. the predictions.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.
- virtual Dtype get_normalizer (LossParameter_NormalizationMode normalization_mode, int valid_count)

Protected Attributes

- shared_ptr< Layer< Dtype > > softmax_layer_

  The internal SoftmaxLayer used to map predictions to a distribution.
- Blob< Dtype > prob_

  prob stores the output probability predictions from the SoftmaxLayer.
- vector< Blob< Dtype > * > softmax_bottom_vec_

  bottom vector holder used in call to the underlying SoftmaxLayer::Forward
- vector< Blob< Dtype > * > softmax_top_vec_

  top vector holder used in call to the underlying SoftmaxLayer::Forward
- bool has_ignore_label_

  Whether to ignore instances with a certain label.
- int ignore_label_

  The label indicating that an instance should be ignored.
- LossParameter_NormalizationMode normalization_

  How to normalize the output loss.
- int **softmax_axis_**
- int **outer_num_**
- int **inner_num_**

5.90.1 Detailed Description

template<typename Dtype>
class caffe::SoftmaxWithLossLayer< Dtype >

Computes the multinomial logistic loss for a one-of-many classification task, passing real-valued predictions through a softmax to get a probability distribution over classes.

This layer should be preferred over separate SoftmaxLayer + MultinomialLogisticLossLayer as its gradient computation is more numerically stable. At test time, this layer can be replaced simply by a SoftmaxLayer.

Parameters

bottom	input Blob vector (length 2) 1. $(N \times C \times H \times W)$ the predictions x, a Blob with values in $[-\infty, +\infty]$ indicating the predicted score for each of the $K = CHW$ classes. This layer maps these scores to a probability distribution over classes using the softmax function $\hat{p}_{nk} = \exp(x_{nk}) / \left[\sum_{k'} \exp(x_{nk'})\right]$ (see SoftmaxLayer). 2. $(N \times 1 \times 1 \times 1)$ the labels l, an integer-valued Blob with values $l_n \in [0, 1, 2, ..., K-1]$ indicating the correct class label among the K classes
top	output Blob vector (length 1) 1. $(1 \times 1 \times 1 \times 1)$ the computed cross-entropy classification loss: $E = \frac{-1}{N} \sum\limits_{n=1}^{N} \log(\hat{p}_{n,l_n})$, for softmax output class probabilites $\hat{p}$

5.90.2 Constructor & Destructor Documentation

5.90.2.1 SoftmaxWithLossLayer()

```
template<typename Dtype >
caffe::SoftmaxWithLossLayer< Dtype >::SoftmaxWithLossLayer (
            const LayerParameter & param )  [inline], [explicit]
```

Parameters

param	provides LossParameter loss_param, with options: • ignore_label (optional) Specify a label value that should be ignored when computing the loss. • normalize (optional, default true) If true, the loss is normalized by the number of (nonignored) labels present; otherwise the loss is simply summed over spatial locations.

5.90.3 Member Function Documentation

5.90.3.1 Backward_cpu()

```
template<typename Dtype >
void caffe::SoftmaxWithLossLayer< Dtype >::Backward_cpu (
            const vector< Blob< Dtype > *> & top,
            const vector< bool > & propagate_down,
            const vector< Blob< Dtype > *> & bottom )  [protected], [virtual]
```

Computes the softmax loss error gradient w.r.t. the predictions.

Gradients cannot be computed with respect to the label inputs (bottom[1]), so this method ignores bottom[1] and requires !propagate_down[1], crashing if propagate_down[1] is set.

Parameters

top	output Blob vector (length 1), providing the error gradient with respect to the outputs 1. $(1 \times 1 \times 1 \times 1)$ This Blob's diff will simply contain the loss_weight$*$ λ, as λ is the coefficient of this layer's output ℓ_i in the overall Net loss $E = \lambda_i \ell_i +$ other loss terms; hence $\frac{\partial E}{\partial \ell_i} = \lambda_i$. ($*$Assuming that this top Blob is not used as a bottom (input) by any other layer of the Net.)
propagate_down	see Layer::Backward. propagate_down[1] must be false as we can't compute gradients with respect to the labels.
bottom	input Blob vector (length 2) 1. $(N \times C \times H \times W)$ the predictions x; Backward computes diff $\frac{\partial E}{\partial x}$ 2. $(N \times 1 \times 1 \times 1)$ the labels – ignored as we can't compute their error gradients

Implements caffe::Layer< Dtype >.

5.90.3.2 ExactNumTopBlobs()

```
template<typename Dtype >
virtual int caffe::SoftmaxWithLossLayer< Dtype >::ExactNumTopBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of top blobs.

Reimplemented from caffe::LossLayer< Dtype >.

5.90.3.3 get_normalizer()

```
template<typename Dtype >
Dtype caffe::SoftmaxWithLossLayer< Dtype >::get_normalizer (
            LossParameter_NormalizationMode normalization_mode,
            int valid_count )  [protected], [virtual]
```

Read the normalization mode parameter and compute the normalizer based on the blob size. If normalization_↩ mode is VALID, the count of valid outputs will be read from valid_count, unless it is -1 in which case all outputs are assumed to be valid.

5.90.3.4 LayerSetUp()

```
template<typename Dtype >
void caffe::SoftmaxWithLossLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::LossLayer< Dtype >.

5.90.3.5 MaxTopBlobs()

```
template<typename Dtype >
virtual int caffe::SoftmaxWithLossLayer< Dtype >::MaxTopBlobs ( ) const  [inline], [virtual]
```

Returns the maximum number of top blobs required by the layer, or -1 if no maximum number is required.

This method should be overridden to return a non-negative value if your layer expects some maximum number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.90.3.6 MinTopBlobs()

```
template<typename Dtype >
virtual int caffe::SoftmaxWithLossLayer< Dtype >::MinTopBlobs ( ) const  [inline], [virtual]
```

Returns the minimum number of top blobs required by the layer, or -1 if no minimum number is required.

This method should be overridden to return a non-negative value if your layer expects some minimum number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.90.3.7 Reshape()

```
template<typename Dtype >
void caffe::SoftmaxWithLossLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Reimplemented from caffe::LossLayer< Dtype >.

The documentation for this class was generated from the following files:

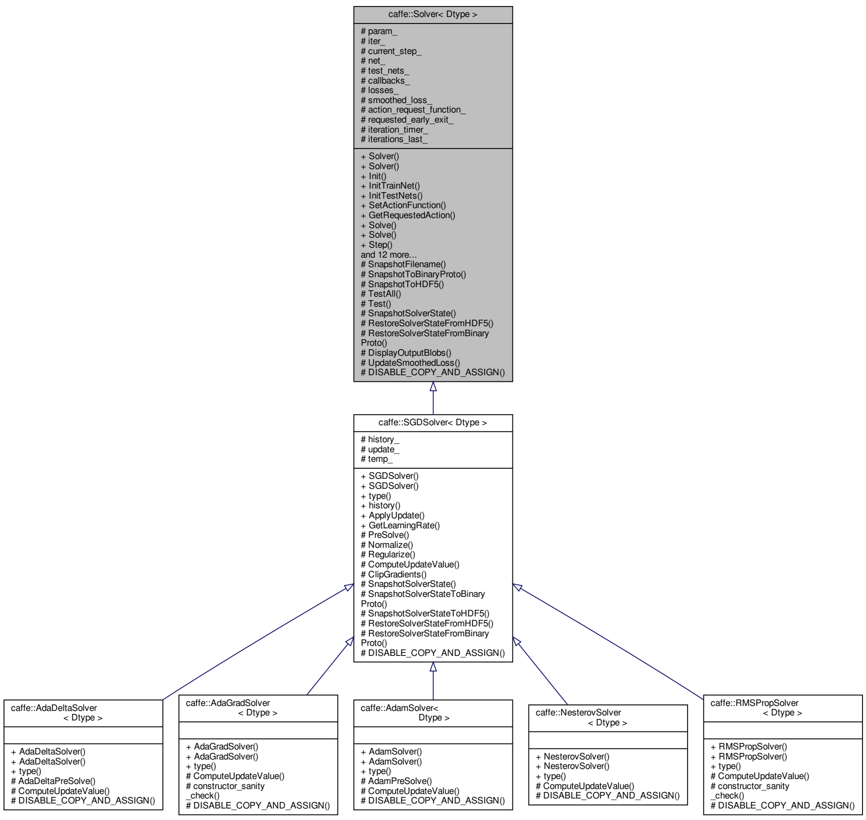
- include/caffe/layers/softmax_loss_layer.hpp
- src/caffe/layers/softmax_loss_layer.cpp

5.91 caffe::Solver< Dtype > Class Template Reference

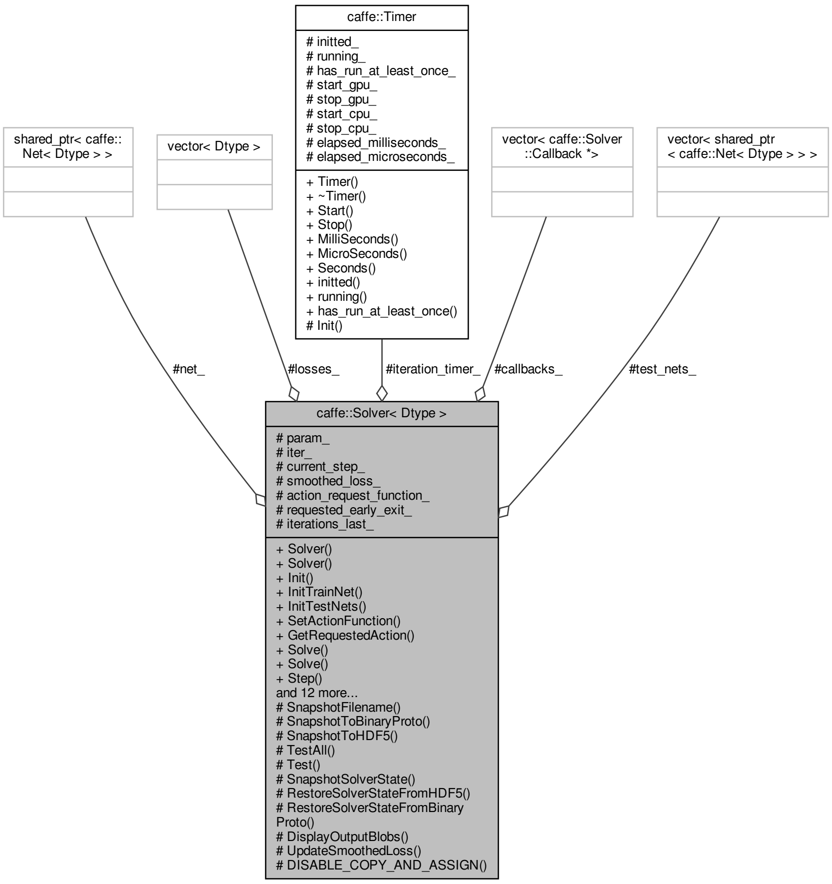
An interface for classes that perform optimization on Nets.

```
#include <solver.hpp>
```

Inheritance diagram for caffe::Solver< Dtype >:



Collaboration diagram for caffe::Solver< Dtype >:



Classes

- class Callback

Public Member Functions

- **Solver** (const SolverParameter &param)
- **Solver** (const string &param_file)
- void **Init** (const SolverParameter &param)
- void **InitTrainNet** ()
- void **InitTestNets** ()
- void **SetActionFunction** (ActionCallback func)
- SolverAction::Enum **GetRequestedAction** ()
- virtual void **Solve** (const char ∗resume_file=NULL)
- void **Solve** (const string &resume_file)
- void **Step** (int iters)
- void **Restore** (const char ∗resume_file)
- void **Snapshot** ()
- const SolverParameter & **param** () const
- shared_ptr< Net< Dtype > > **net** ()

- const vector< shared_ptr< Net< Dtype > > > & **test_nets** ()
- int **iter** () const
- const vector< Callback * > & **callbacks** () const
- void **add_callback** (Callback *value)
- void **CheckSnapshotWritePermissions** ()
- virtual const char * type () const

  Returns the solver type.
- virtual void **ApplyUpdate** ()=0

Protected Member Functions

- string **SnapshotFilename** (const string &extension)
- string **SnapshotToBinaryProto** ()
- string **SnapshotToHDF5** ()
- void **TestAll** ()
- void **Test** (const int test_net_id=0)
- virtual void **SnapshotSolverState** (const string &model_filename)=0
- virtual void **RestoreSolverStateFromHDF5** (const string &state_file)=0
- virtual void **RestoreSolverStateFromBinaryProto** (const string &state_file)=0
- void **DisplayOutputBlobs** (const int net_id)
- void **UpdateSmoothedLoss** (Dtype loss, int start_iter, int average_loss)
- **DISABLE_COPY_AND_ASSIGN** (Solver)

Protected Attributes

- SolverParameter **param_**
- int **iter_**
- int **current_step_**
- shared_ptr< Net< Dtype > > **net_**
- vector< shared_ptr< Net< Dtype > > > **test_nets_**
- vector< Callback * > **callbacks_**
- vector< Dtype > **losses_**
- Dtype **smoothed_loss_**
- ActionCallback **action_request_function_**
- bool **requested_early_exit_**
- Timer **iteration_timer_**
- float **iterations_last_**

5.91.1 Detailed Description

template<typename Dtype>
class caffe::Solver< Dtype >

An interface for classes that perform optimization on Nets.

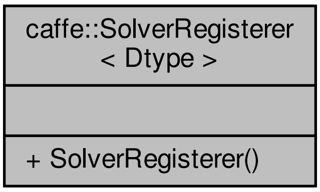
Requires implementation of ApplyUpdate to compute a parameter update given the current state of the Net parameters.

The documentation for this class was generated from the following files:

- include/caffe/solver.hpp
- src/caffe/solver.cpp

5.92 caffe::SolverRegisterer< Dtype > Class Template Reference

Collaboration diagram for caffe::SolverRegisterer< Dtype >:



Public Member Functions

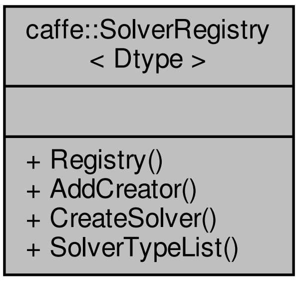
- **SolverRegisterer** (const string &type, Solver< Dtype > ∗(∗creator)(const SolverParameter &))

The documentation for this class was generated from the following file:

- include/caffe/solver_factory.hpp

5.93 caffe::SolverRegistry< Dtype > Class Template Reference

Collaboration diagram for caffe::SolverRegistry< Dtype >:



Public Types

- typedef Solver< Dtype > ∗(∗ **Creator**) (const SolverParameter &)
- typedef std::map< string, Creator > **CreatorRegistry**

Static Public Member Functions

- static CreatorRegistry & **Registry** ()
- static void **AddCreator** (const string &type, Creator creator)
- static Solver< Dtype > ∗ **CreateSolver** (const SolverParameter &param)
- static vector< string > **SolverTypeList** ()

The documentation for this class was generated from the following file:

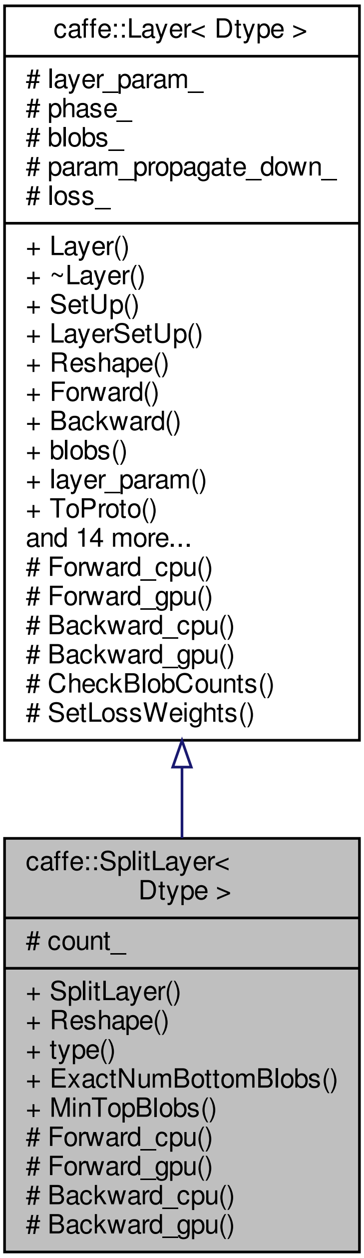
- include/caffe/solver_factory.hpp

5.94 caffe::SplitLayer< Dtype > Class Template Reference

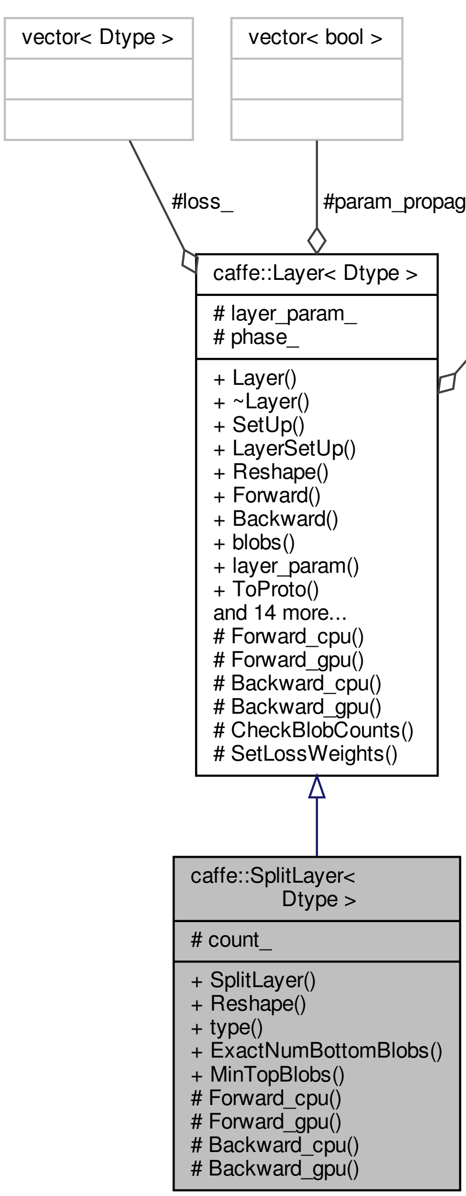
Creates a "split" path in the network by copying the bottom Blob into multiple top Blobs to be used by multiple consuming layers.

```
#include <split_layer.hpp>
```

Inheritance diagram for caffe::SplitLayer< Dtype >:



Collaboration diagram for caffe::SplitLayer< Dtype >:



Public Member Functions

- **SplitLayer** (const LayerParameter &param)
- virtual void Reshape (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

    Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.
- virtual const char ∗ type () const

    Returns the layer type.
- virtual int ExactNumBottomBlobs () const

    Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.
- virtual int MinTopBlobs () const

    Returns the minimum number of top blobs required by the layer, or -1 if no minimum number is required.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

    Using the CPU device, compute the layer output.
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

    Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

    Using the CPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

    Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Protected Attributes

- int **count_**

5.94.1 Detailed Description

template<typename Dtype>
class caffe::SplitLayer< Dtype >

Creates a "split" path in the network by copying the bottom Blob into multiple top Blobs to be used by multiple consuming layers.

TODO(dox): thorough documentation for Forward, Backward, and proto params.

5.94.2 Member Function Documentation

5.94.2.1 ExactNumBottomBlobs()

```
template<typename Dtype >
virtual int caffe::SplitLayer< Dtype >::ExactNumBottomBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.94.2.2 MinTopBlobs()

```
template<typename Dtype >
virtual int caffe::SplitLayer< Dtype >::MinTopBlobs ( ) const [inline], [virtual]
```

Returns the minimum number of top blobs required by the layer, or -1 if no minimum number is required.

This method should be overridden to return a non-negative value if your layer expects some minimum number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.94.2.3 Reshape()

```
template<typename Dtype >
void caffe::SplitLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

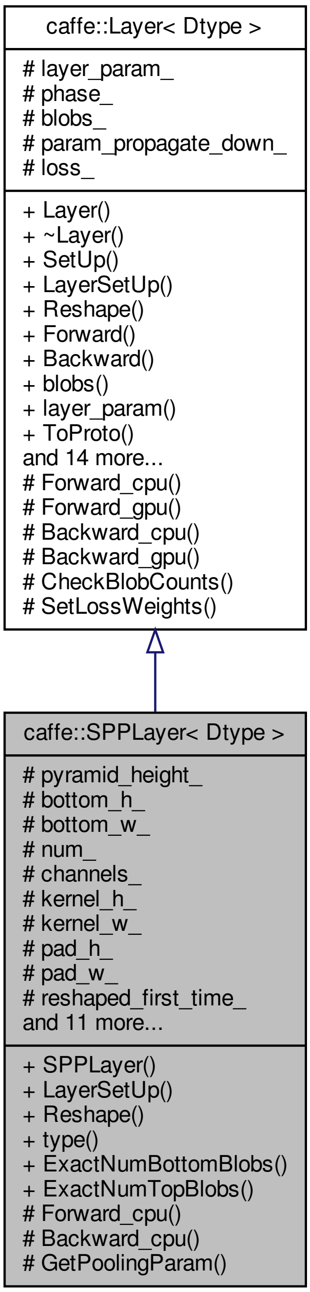
- include/caffe/layers/split_layer.hpp
- src/caffe/layers/split_layer.cpp

5.95 caffe::SPPLayer< Dtype > Class Template Reference

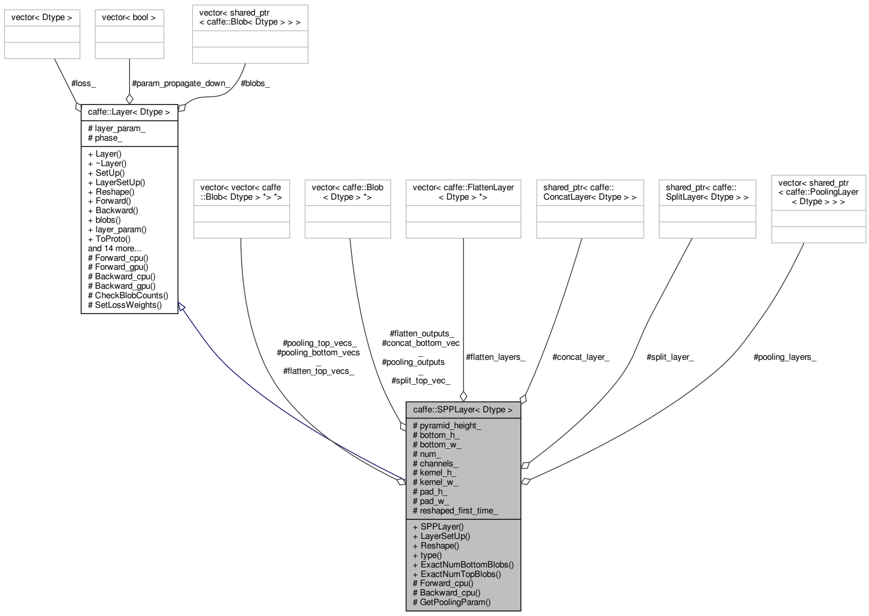
Does spatial pyramid pooling on the input image by taking the max, average, etc. within regions so that the result vector of different sized images are of the same size.

```
#include <spp_layer.hpp>
```

Inheritance diagram for caffe::SPPLayer< Dtype >:



Collaboration diagram for caffe::SPPLayer< Dtype >:



Public Member Functions

- **SPPLayer** (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

    Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void Reshape (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

    Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.
- virtual const char ∗ type () const

    Returns the layer type.
- virtual int ExactNumBottomBlobs () const

    Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.
- virtual int ExactNumTopBlobs () const

    Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

    Using the CPU device, compute the layer output.
- virtual void Backward_cpu (const vector< Blob< Dtype > ∗> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > ∗> &bottom)

    Using the CPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true.
- virtual LayerParameter **GetPoolingParam** (const int pyramid_level, const int bottom_h, const int bottom_w, const SPPParameter spp_param)

Protected Attributes

- int **pyramid_height_**
- int **bottom_h_**
- int **bottom_w_**
- int **num_**
- int **channels_**
- int **kernel_h_**
- int **kernel_w_**
- int **pad_h_**
- int **pad_w_**
- bool **reshaped_first_time_**
- shared_ptr< SplitLayer< Dtype > > split_layer_

  the internal Split layer that feeds the pooling layers
- vector< Blob< Dtype > ∗ > split_top_vec_

  top vector holder used in call to the underlying SplitLayer::Forward
- vector< vector< Blob< Dtype > ∗ > ∗ > pooling_bottom_vecs_

  bottom vector holder used in call to the underlying PoolingLayer::Forward
- vector< shared_ptr< PoolingLayer< Dtype > > > pooling_layers_

  the internal Pooling layers of different kernel sizes
- vector< vector< Blob< Dtype > ∗ > ∗ > pooling_top_vecs_

  top vector holders used in call to the underlying PoolingLayer::Forward
- vector< Blob< Dtype > ∗ > pooling_outputs_

  pooling_outputs stores the outputs of the PoolingLayers
- vector< FlattenLayer< Dtype > ∗ > flatten_layers_

  the internal Flatten layers that the Pooling layers feed into
- vector< vector< Blob< Dtype > ∗ > ∗ > flatten_top_vecs_

  top vector holders used in call to the underlying FlattenLayer::Forward
- vector< Blob< Dtype > ∗ > flatten_outputs_

  flatten_outputs stores the outputs of the FlattenLayers
- vector< Blob< Dtype > ∗ > concat_bottom_vec_

  bottom vector holder used in call to the underlying ConcatLayer::Forward
- shared_ptr< ConcatLayer< Dtype > > concat_layer_

  the internal Concat layers that the Flatten layers feed into

5.95.1 Detailed Description

template<typename Dtype>
class caffe::SPPLayer< Dtype >

Does spatial pyramid pooling on the input image by taking the max, average, etc. within regions so that the result vector of different sized images are of the same size.

5.95.2 Member Function Documentation

5.95.2.1 ExactNumBottomBlobs()

```
template<typename Dtype >
virtual int caffe::SPPLayer< Dtype >::ExactNumBottomBlobs ( ) const [inline], [virtual]
```

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.95.2.2 ExactNumTopBlobs()

```
template<typename Dtype >
virtual int caffe::SPPLayer< Dtype >::ExactNumTopBlobs ( ) const [inline], [virtual]
```

Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.95.2.3 LayerSetUp()

```
template<typename Dtype >
void caffe::SPPLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

5.95.2.4 Reshape()

```
template<typename Dtype >
void caffe::SPPLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

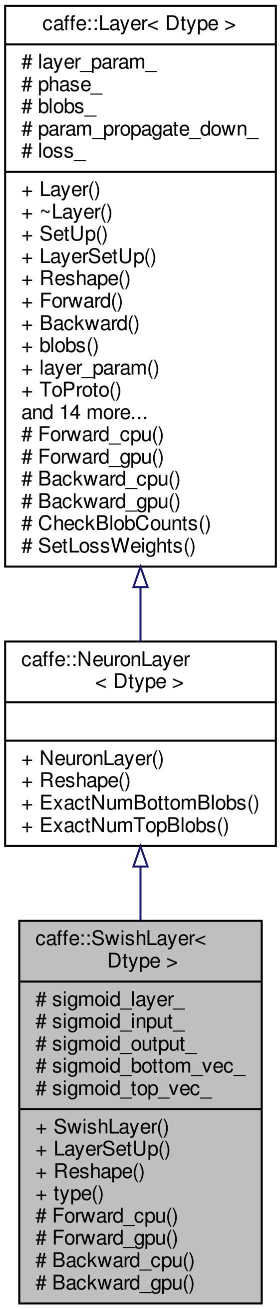
- include/caffe/layers/spp_layer.hpp
- src/caffe/layers/spp_layer.cpp

5.96 caffe::SwishLayer< Dtype > Class Template Reference

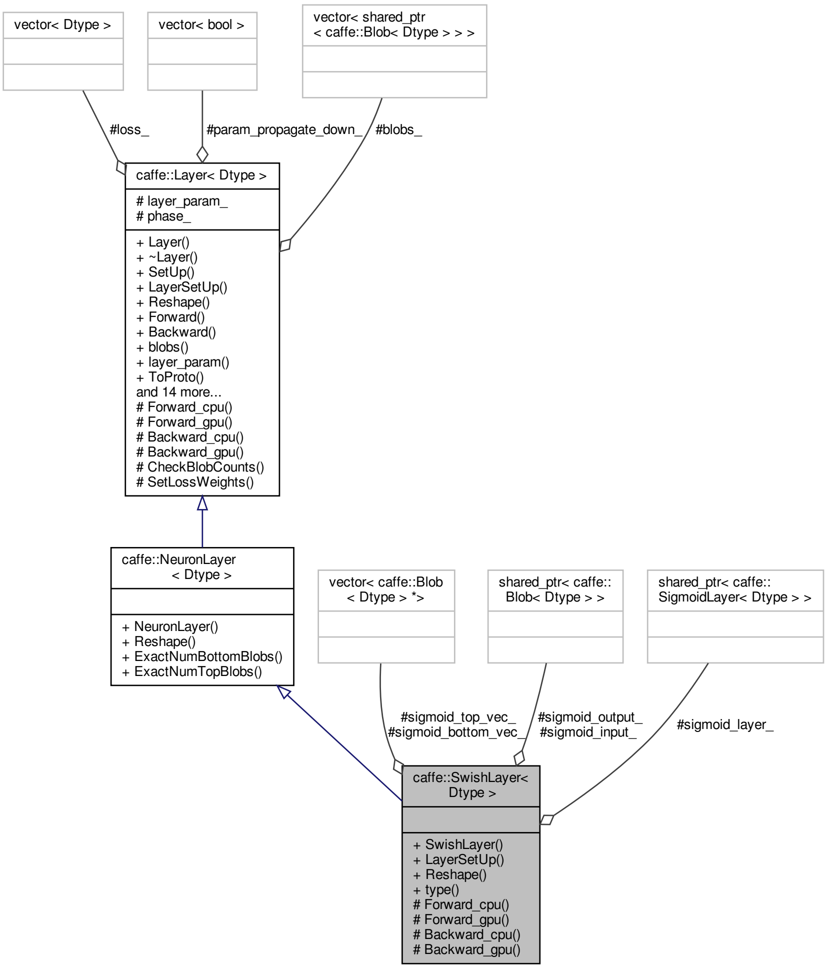
Swish non-linearity $y = x \sigma (\beta x)$. A novel activation function that tends to work better than ReLU [1].

```
#include <swish_layer.hpp>
```

Inheritance diagram for caffe::SwishLayer< Dtype >:



Collaboration diagram for caffe::SwishLayer< Dtype >:



Public Member Functions

- SwishLayer (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual void Reshape (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.
- virtual const char * type () const

  Returns the layer type.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.

- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Computes the error gradient w.r.t. the sigmoid inputs.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Protected Attributes

- shared_ptr< SigmoidLayer< Dtype > > sigmoid_layer_

  The internal SigmoidLayer.
- shared_ptr< Blob< Dtype > > sigmoid_input_

  sigmoid_input_ stores the input of the SigmoidLayer.
- shared_ptr< Blob< Dtype > > sigmoid_output_

  sigmoid_output_ stores the output of the SigmoidLayer.
- vector< Blob< Dtype > * > sigmoid_bottom_vec_

  bottom vector holder to call the underlying SigmoidLayer::Forward
- vector< Blob< Dtype > * > sigmoid_top_vec_

  top vector holder to call the underlying SigmoidLayer::Forward

5.96.1 Detailed Description

template<typename Dtype>
class caffe::SwishLayer< Dtype >

Swish non-linearity $y = x\sigma(\beta x)$. A novel activation function that tends to work better than ReLU [1].

[1] Prajit Ramachandran, Barret Zoph, Quoc V. Le. "Searching for Activation Functions". arXiv preprint arXiv↩
:1710.05941v2 (2017).

5.96.2 Constructor & Destructor Documentation

5.96.2.1 SwishLayer()

```
template<typename Dtype >
caffe::SwishLayer< Dtype >::SwishLayer (
            const LayerParameter & param )  [inline], [explicit]
```

Parameters

param	provides SwishParameter swish_param, with SwishLayer options: • beta (**optional**, default 1). the value β in the $y = x\sigma(\beta x)$.

5.96.3 Member Function Documentation

5.96.3.1 Backward_cpu()

```
template<typename Dtype >
void caffe::SwishLayer< Dtype >::Backward_cpu (
            const vector< Blob< Dtype > *> & top,
            const vector< bool > & propagate_down,
            const vector< Blob< Dtype > *> & bottom )  [protected], [virtual]
```

Computes the error gradient w.r.t. the sigmoid inputs.

Parameters

top	output Blob vector (length 1), providing the error gradient with respect to the outputs 1. $(N \times C \times H \times W)$ containing error gradients $\frac{\partial E}{\partial y}$ with respect to computed outputs y
propagate_down	see Layer::Backward.
bottom	input Blob vector (length 1) 1. $(N \times C \times H \times W)$ the inputs x; Backward fills their diff with gradients $\frac{\partial E}{\partial x} = \frac{\partial E}{\partial y}(\beta y + \sigma(\beta x)(1 - \beta y))$ if propagate_down[0]

Implements caffe::Layer< Dtype >.

5.96.3.2 Forward_cpu()

```
template<typename Dtype >
void caffe::SwishLayer< Dtype >::Forward_cpu (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [protected], [virtual]
```

Parameters

bottom	input Blob vector (length 1) 1. $(N \times C \times H \times W)$ the inputs x
top	output Blob vector (length 1) 1. $(N \times C \times H \times W)$ the computed outputs $y = x\sigma(\beta x)$.

Implements caffe::Layer< Dtype >.

5.96.3.3 LayerSetUp()

```
template<typename Dtype >
void caffe::SwishLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

5.96.3.4 Reshape()

```
template<typename Dtype >
void caffe::SwishLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

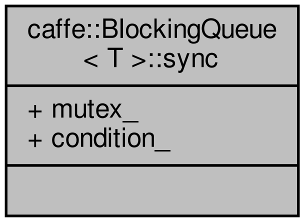
Reimplemented from caffe::NeuronLayer< Dtype >.

The documentation for this class was generated from the following files:

- include/caffe/layers/swish_layer.hpp
- src/caffe/layers/swish_layer.cpp

5.97 caffe::BlockingQueue< T >::sync Class Reference

Collaboration diagram for caffe::BlockingQueue< T >::sync:



Public Attributes

- boost::mutex **mutex_**
- boost::condition_variable **condition_**

The documentation for this class was generated from the following file:

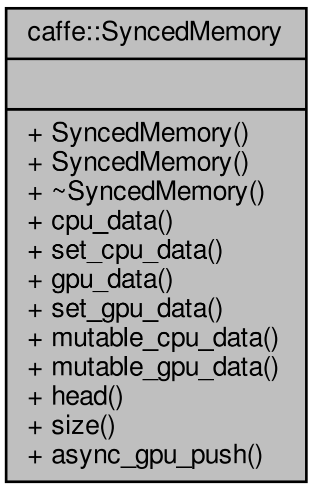
- src/caffe/util/blocking_queue.cpp

5.98 caffe::SyncedMemory Class Reference

Manages memory allocation and synchronization between the host (CPU) and device (GPU).

`#include <syncedmem.hpp>`

Collaboration diagram for caffe::SyncedMemory:



Public Types

- enum **SyncedHead** { **UNINITIALIZED**, **HEAD_AT_CPU**, **HEAD_AT_GPU**, **SYNCED** }

Public Member Functions

- **SyncedMemory** (size_t size)
- const void * **cpu_data** ()
- void **set_cpu_data** (void *data)
- const void * **gpu_data** ()
- void **set_gpu_data** (void *data)
- void * **mutable_cpu_data** ()
- void * **mutable_gpu_data** ()
- SyncedHead **head** () const
- size_t **size** () const
- void **async_gpu_push** (const cudaStream_t &stream)

5.98.1 Detailed Description

Manages memory allocation and synchronization between the host (CPU) and device (GPU).

TODO(dox): more thorough description.

The documentation for this class was generated from the following files:

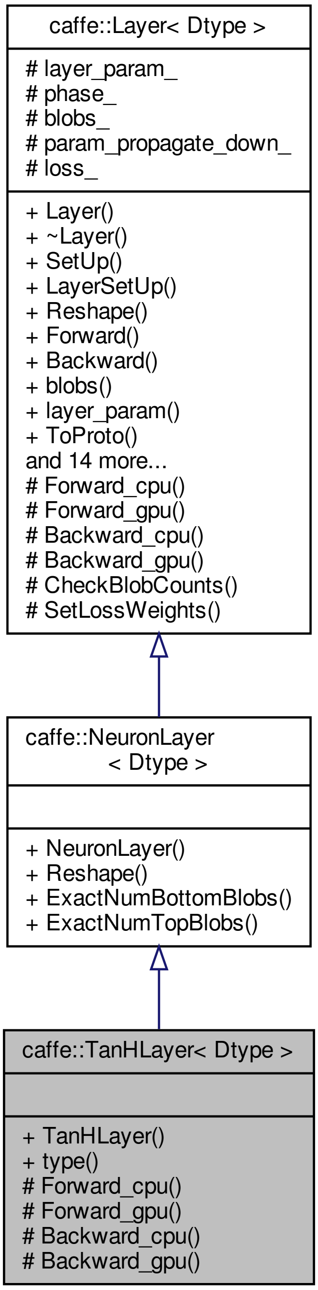
- include/caffe/syncedmem.hpp
- src/caffe/syncedmem.cpp

5.99 caffe::TanHLayer< Dtype > Class Template Reference

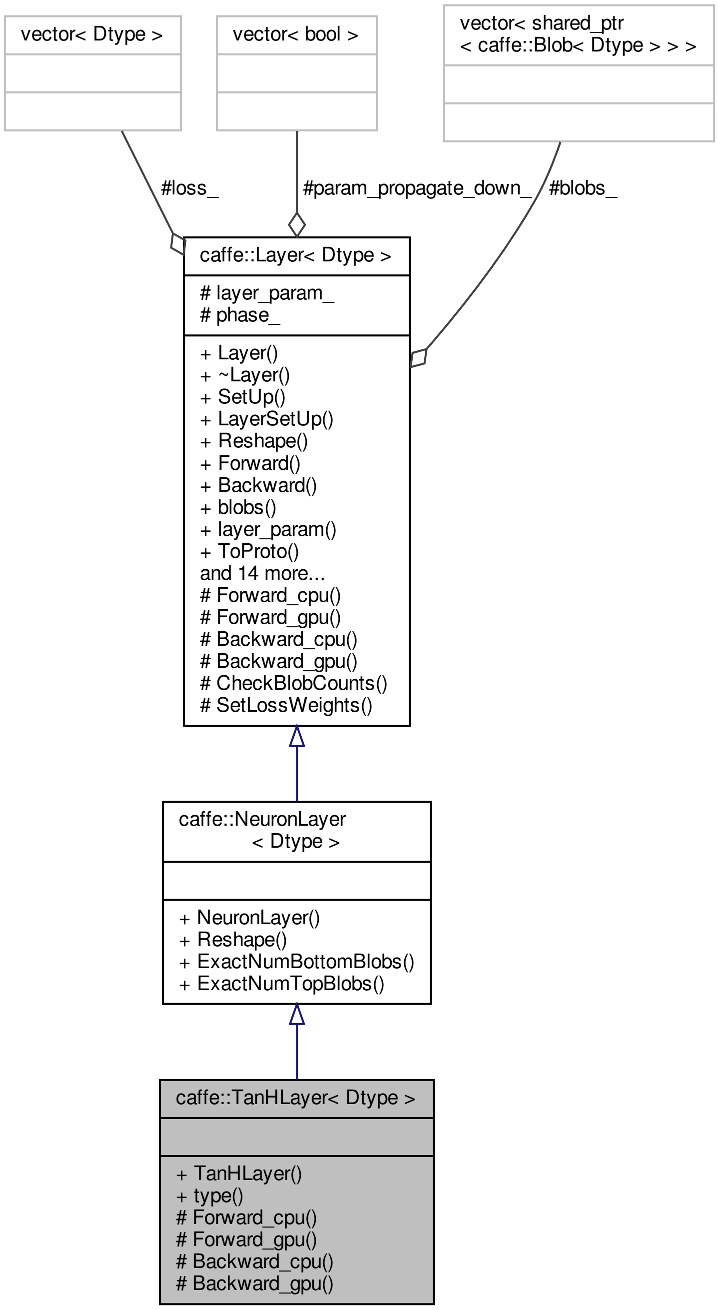
TanH hyperbolic tangent non-linearity $y = \frac{\exp(2x)-1}{\exp(2x)+1}$, popular in auto-encoders.

`#include <tanh_layer.hpp>`

Inheritance diagram for caffe::TanHLayer< Dtype >:



Collaboration diagram for caffe::TanHLayer< Dtype >:



Public Member Functions

- **TanHLayer** (const LayerParameter &param)
- virtual const char ∗ type () const

    Returns the layer type.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Computes the error gradient w.r.t. the sigmoid inputs.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Additional Inherited Members

5.99.1 Detailed Description

template<typename Dtype>
class caffe::TanHLayer< Dtype >

TanH hyperbolic tangent non-linearity $y = \frac{\exp(2x)-1}{\exp(2x)+1}$, popular in auto-encoders.

Note that the gradient vanishes as the values move away from 0. The ReLULayer is often a better choice for this reason.

5.99.2 Member Function Documentation

5.99.2.1 Backward_cpu()

```
template<typename Dtype >
void caffe::TanHLayer< Dtype >::Backward_cpu (
            const vector< Blob< Dtype > *> & top,
            const vector< bool > & propagate_down,
            const vector< Blob< Dtype > *> & bottom ) [protected], [virtual]
```

Computes the error gradient w.r.t. the sigmoid inputs.

Parameters

top	output Blob vector (length 1), providing the error gradient with respect to the outputs 1. $(N \times C \times H \times W)$ containing error gradients $\frac{\partial E}{\partial y}$ with respect to computed outputs y
propagate_down	see Layer::Backward.
bottom	input Blob vector (length 1) 1. $(N \times C \times H \times W)$ the inputs x; Backward fills their diff with gradients $\frac{\partial E}{\partial x} = \frac{\partial E}{\partial y}\left(1 - \left[\frac{\exp(2x)-1}{exp(2x)+1}\right]^2\right) = \frac{\partial E}{\partial y}(1-y^2)$ if propagate_down[0]

Implements caffe::Layer< Dtype >.

5.99.2.2 Forward_cpu()

```
template<typename Dtype >
void caffe::TanHLayer< Dtype >::Forward_cpu (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top )  [protected], [virtual]
```

Parameters

bottom	input Blob vector (length 1) 1. $(N \times C \times H \times W)$ the inputs x
top	output Blob vector (length 1) 1. $(N \times C \times H \times W)$ the computed outputs $y = \frac{\exp(2x)-1}{\exp(2x)+1}$

Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

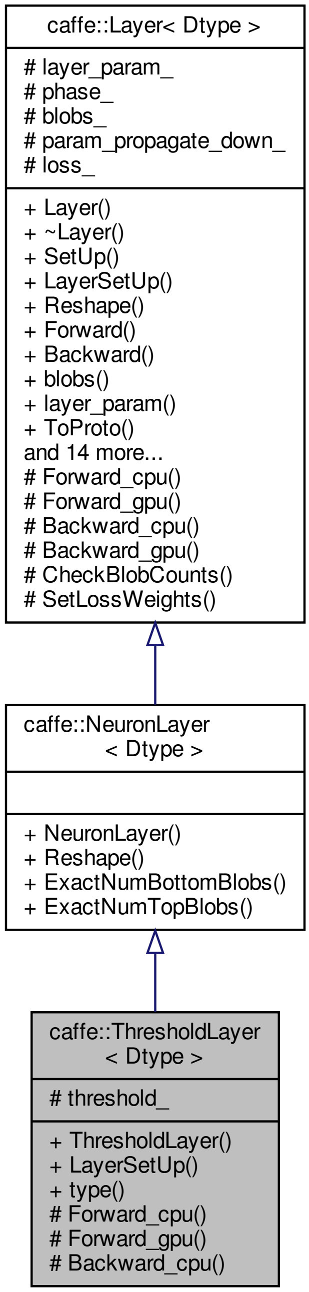
- include/caffe/layers/tanh_layer.hpp
- src/caffe/layers/tanh_layer.cpp

5.100 caffe::ThresholdLayer< Dtype > Class Template Reference

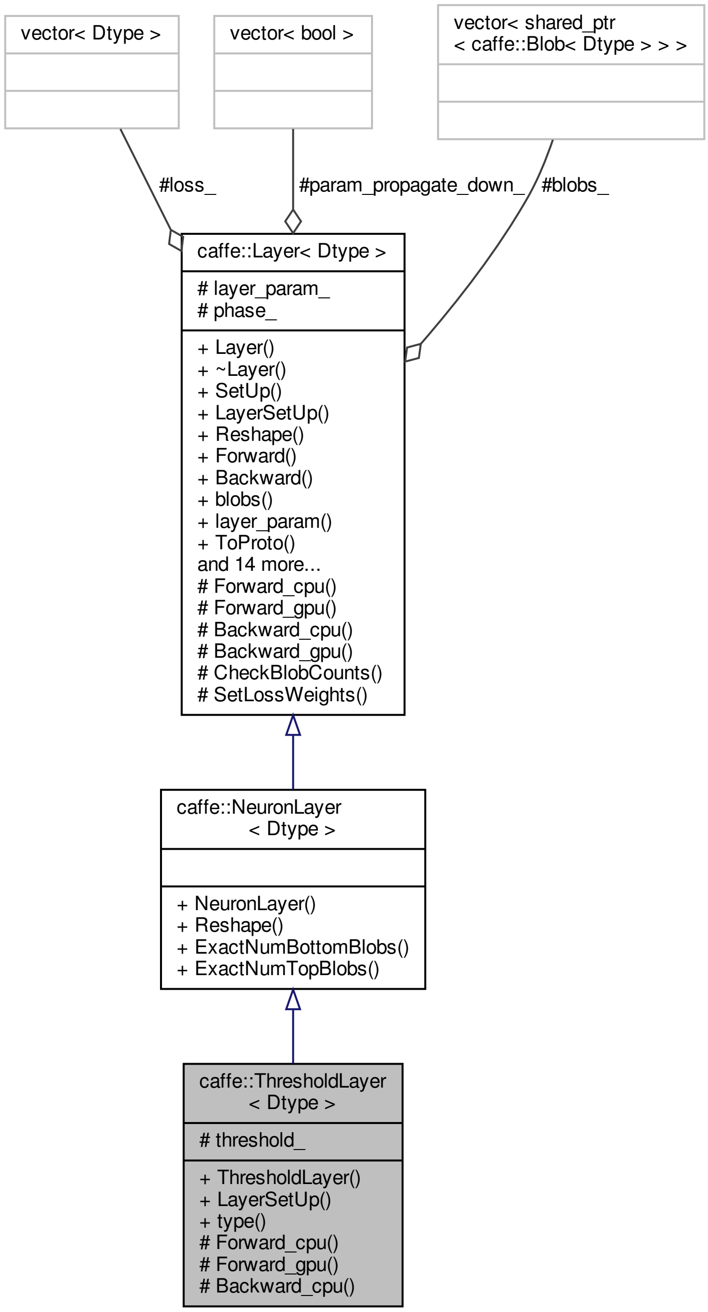
Tests whether the input exceeds a threshold: outputs 1 for inputs above threshold; 0 otherwise.

```
#include <threshold_layer.hpp>
```

Inheritance diagram for caffe::ThresholdLayer< Dtype >:



Collaboration diagram for caffe::ThresholdLayer< Dtype >:



Public Member Functions

- ThresholdLayer (const LayerParameter &param)
- virtual void LayerSetUp (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Does layer-specific setup: your layer should implement this function as well as Reshape.
- virtual const char ∗ type () const

  Returns the layer type.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Not implemented (non-differentiable function)

Protected Attributes

- Dtype **threshold_**

5.100.1 Detailed Description

template<typename Dtype>
class caffe::ThresholdLayer< Dtype >

Tests whether the input exceeds a threshold: outputs 1 for inputs above threshold; 0 otherwise.

5.100.2 Constructor & Destructor Documentation

5.100.2.1 ThresholdLayer()

```
template<typename Dtype >
caffe::ThresholdLayer< Dtype >::ThresholdLayer (
            const LayerParameter & param )  [inline], [explicit]
```

Parameters

param	provides ThresholdParameter threshold_param, with ThresholdLayer options: • threshold (**optional**, default 0). the threshold value t to which the input values are compared.

5.100.3 Member Function Documentation

5.100.3.1 Forward_cpu()

```
template<typename Dtype >
void caffe::ThresholdLayer< Dtype >::Forward_cpu (
```

```
const vector< Blob< Dtype > *> & bottom,
const vector< Blob< Dtype > *> & top ) [protected], [virtual]
```

Parameters

bottom	input Blob vector (length 1) 1. $(N \times C \times H \times W)$ the inputs x
top	output Blob vector (length 1) 1. $(N \times C \times H \times W)$ the computed outputs $y = \begin{cases} 0 & \text{if } x \leq t \\ 1 & \text{if } x > t \end{cases}$

Implements caffe::Layer< Dtype >.

5.100.3.2 LayerSetUp()

```
template<typename Dtype >
void caffe::ThresholdLayer< Dtype >::LayerSetUp (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [virtual]
```

Does layer-specific setup: your layer should implement this function as well as Reshape.

Parameters

bottom	the preshaped input blobs, whose data fields store the input data for this layer
top	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the `layer_param_`. Setting up the shapes of top blobs and internal buffers should be done in `Reshape`, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

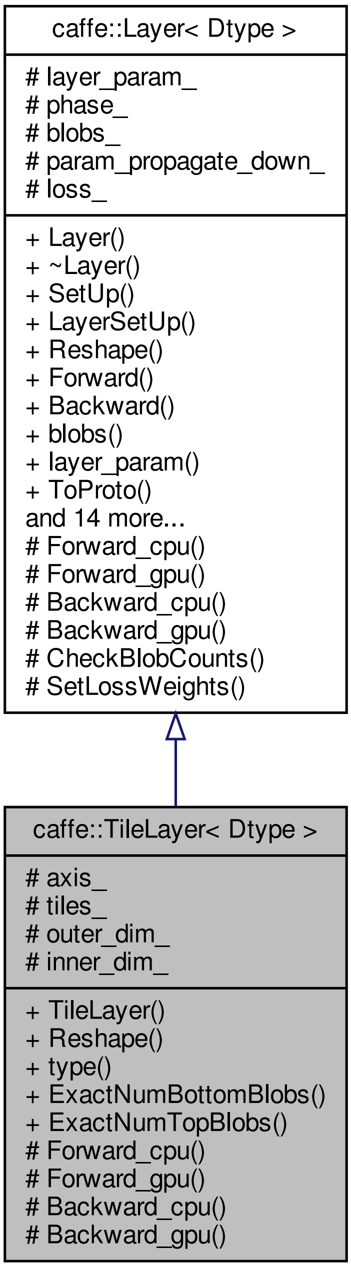
- include/caffe/layers/threshold_layer.hpp
- src/caffe/layers/threshold_layer.cpp

5.101 caffe::TileLayer< Dtype > Class Template Reference

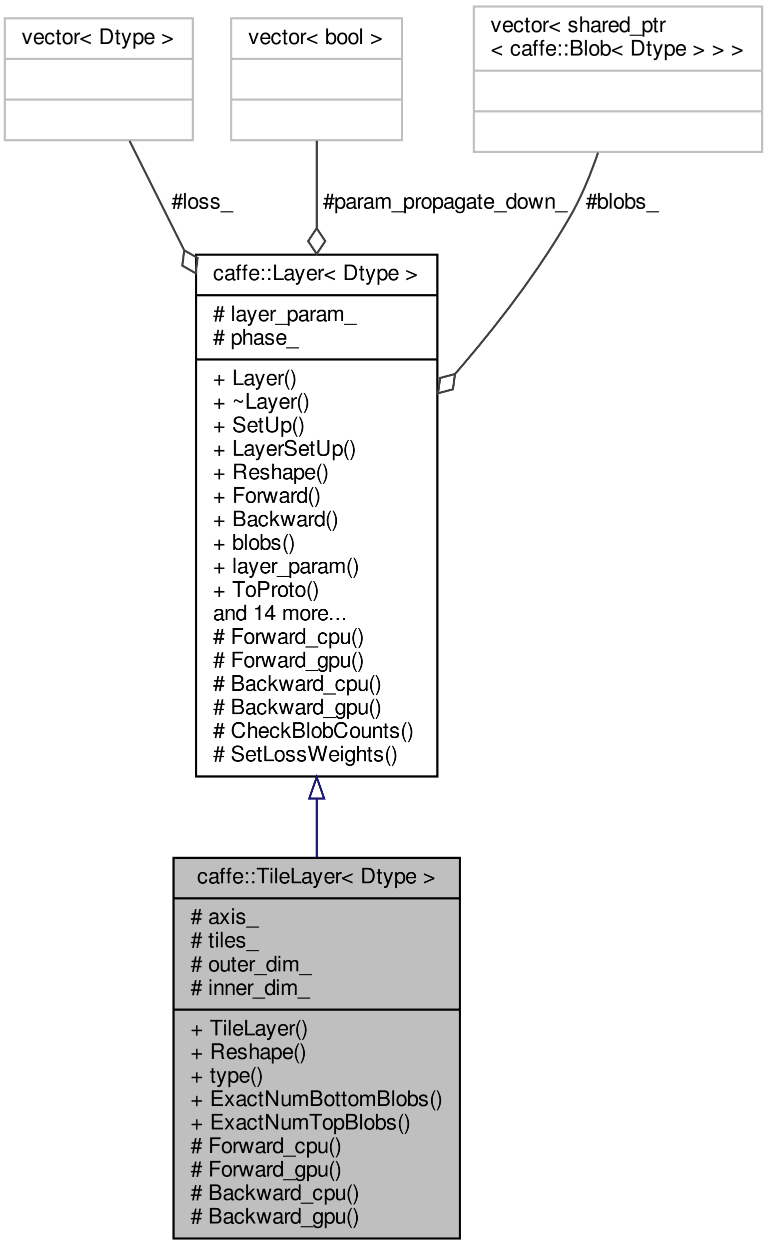
Copy a Blob along specified dimensions.

`#include <tile_layer.hpp>`

Inheritance diagram for caffe::TileLayer< Dtype >:



Collaboration diagram for caffe::TileLayer< Dtype >:



Public Member Functions

- **TileLayer** (const LayerParameter &param)
- virtual void Reshape (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)

  Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.
- virtual const char ∗ type () const

  Returns the layer type.
- virtual int ExactNumBottomBlobs () const

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

- virtual int ExactNumTopBlobs () const

  Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

Protected Member Functions

- virtual void Forward_cpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the CPU device, compute the layer output.
- virtual void Forward_gpu (const vector< Blob< Dtype > *> &bottom, const vector< Blob< Dtype > *> &top)

  Using the GPU device, compute the layer output. Fall back to Forward_cpu() if unavailable.
- virtual void Backward_cpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the CPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true.
- virtual void Backward_gpu (const vector< Blob< Dtype > *> &top, const vector< bool > &propagate_down, const vector< Blob< Dtype > *> &bottom)

  Using the GPU device, compute the gradients for any parameters and for the bottom blobs if propagate_down is true. Fall back to Backward_cpu() if unavailable.

Protected Attributes

- unsigned int **axis_**
- unsigned int **tiles_**
- unsigned int **outer_dim_**
- unsigned int **inner_dim_**

5.101.1 Detailed Description

template<typename Dtype>
class caffe::TileLayer< Dtype >

Copy a Blob along specified dimensions.

5.101.2 Member Function Documentation

5.101.2.1 ExactNumBottomBlobs()

```
template<typename Dtype >
virtual int caffe::TileLayer< Dtype >::ExactNumBottomBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.101.2.2 ExactNumTopBlobs()

```
template<typename Dtype >
virtual int caffe::TileLayer< Dtype >::ExactNumTopBlobs ( ) const [inline], [virtual]
```

Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

5.101.2.3 Reshape()

```
template<typename Dtype >
void caffe::TileLayer< Dtype >::Reshape (
            const vector< Blob< Dtype > *> & bottom,
            const vector< Blob< Dtype > *> & top ) [virtual]
```

Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

Parameters

bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

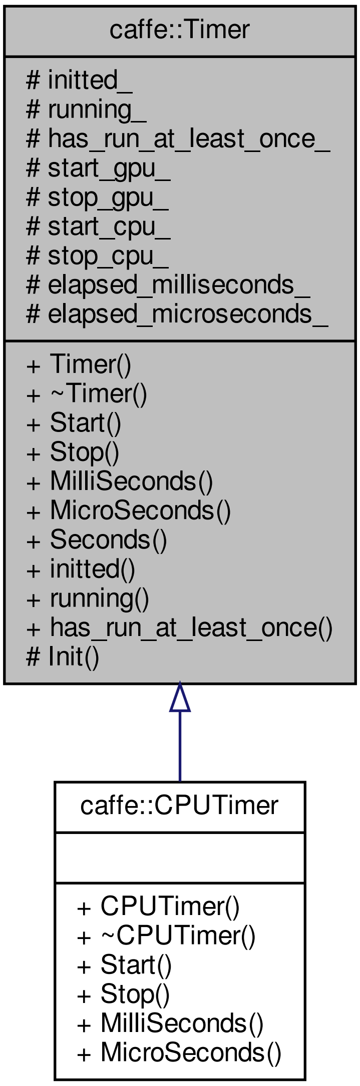
Implements caffe::Layer< Dtype >.

The documentation for this class was generated from the following files:

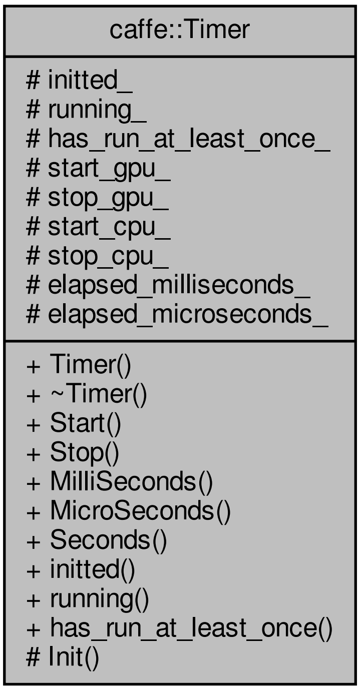
- include/caffe/layers/tile_layer.hpp
- src/caffe/layers/tile_layer.cpp

5.102 caffe::Timer Class Reference

Inheritance diagram for caffe::Timer:



Collaboration diagram for caffe::Timer:



Public Member Functions

- virtual void **Start** ()
- virtual void **Stop** ()
- virtual float **MilliSeconds** ()
- virtual float **MicroSeconds** ()
- virtual float **Seconds** ()
- bool **initted** ()
- bool **running** ()
- bool **has_run_at_least_once** ()

Protected Member Functions

- void **Init** ()

Protected Attributes

- bool **initted_**
- bool **running_**
- bool **has_run_at_least_once_**
- cudaEvent_t **start_gpu_**

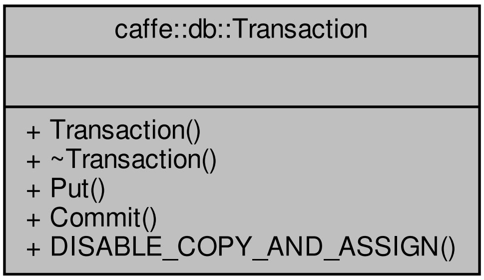
- cudaEvent_t **stop_gpu_**
- boost::posix_time::ptime **start_cpu_**
- boost::posix_time::ptime **stop_cpu_**
- float **elapsed_milliseconds_**
- float **elapsed_microseconds_**

The documentation for this class was generated from the following files:

- include/caffe/util/benchmark.hpp
- src/caffe/util/benchmark.cpp

5.103 caffe::db::Transaction Class Reference

Collaboration diagram for caffe::db::Transaction:



Public Member Functions

- virtual void **Put** (const string &key, const string &value)=0
- virtual void **Commit** ()=0
- **DISABLE_COPY_AND_ASSIGN** (Transaction)

The documentation for this class was generated from the following file:

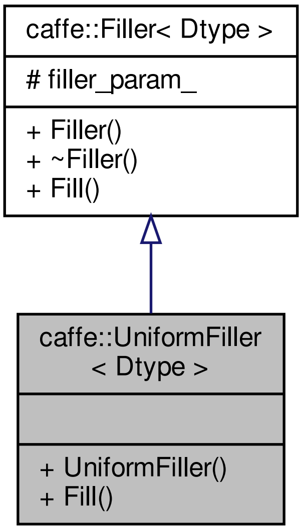
- include/caffe/util/db.hpp

5.104 caffe::UniformFiller< Dtype > Class Template Reference

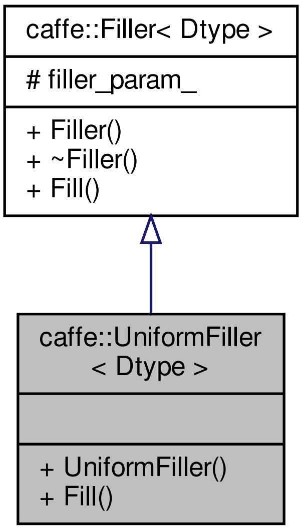
Fills a Blob with uniformly distributed values $x \sim U(a, b)$.

`#include <filler.hpp>`

Inheritance diagram for caffe::UniformFiller< Dtype >:



Collaboration diagram for caffe::UniformFiller< Dtype >:



Public Member Functions

- **UniformFiller** (const FillerParameter &param)
- virtual void **Fill** (Blob$<$ Dtype $>$ $*$blob)

Additional Inherited Members

5.104.1 Detailed Description

template$<$typename Dtype$>$
class caffe::UniformFiller$<$ Dtype $>$

Fills a Blob with uniformly distributed values $x \sim U(a, b)$.

The documentation for this class was generated from the following file:

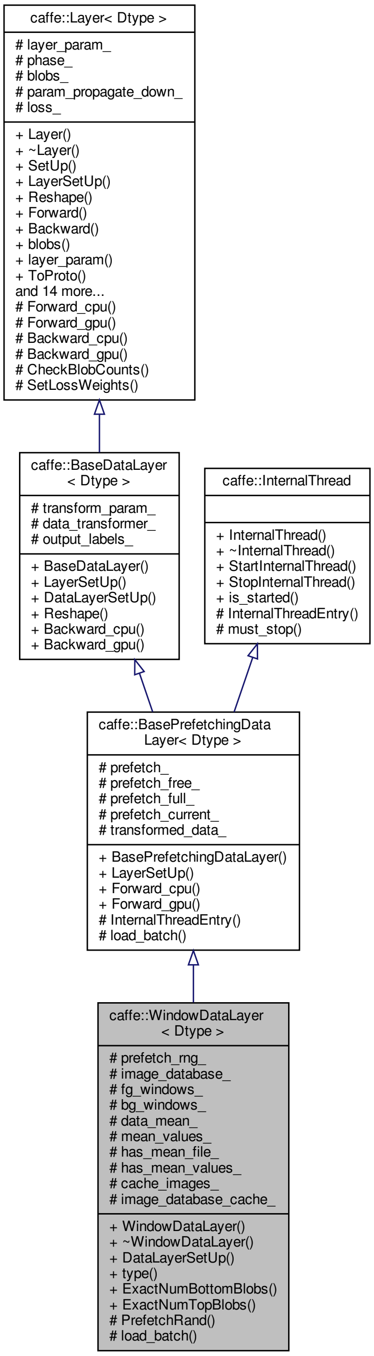
- include/caffe/filler.hpp

5.105 caffe::WindowDataLayer$<$ Dtype $>$ Class Template Reference

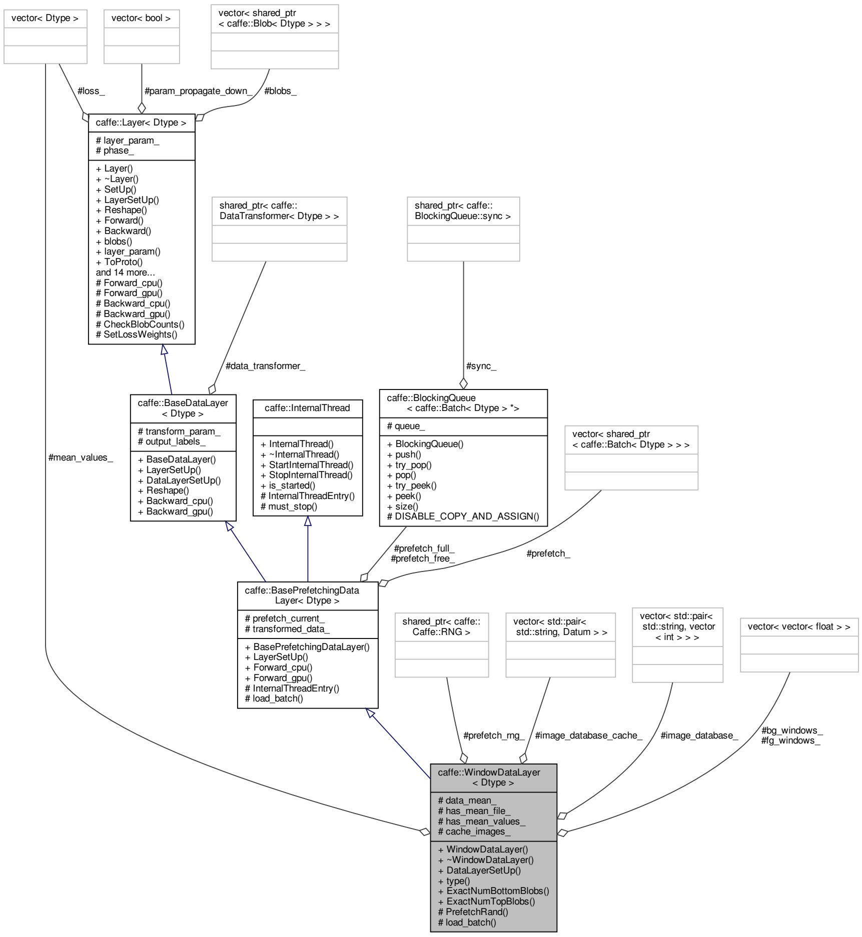
Provides data to the Net from windows of images files, specified by a window data file. This layer is *DEPRECATED* and only kept for archival purposes for use by the original R-CNN.

```
#include <window_data_layer.hpp>
```

Inheritance diagram for caffe::WindowDataLayer< Dtype >:



Collaboration diagram for caffe::WindowDataLayer< Dtype >:



Public Member Functions

- **WindowDataLayer** (const LayerParameter &param)
- virtual void **DataLayerSetUp** (const vector< Blob< Dtype > ∗> &bottom, const vector< Blob< Dtype > ∗> &top)
- virtual const char ∗ type () const

  Returns the layer type.
- virtual int ExactNumBottomBlobs () const

  Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.
- virtual int ExactNumTopBlobs () const

  Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

Protected Types

- enum **WindowField** {
  IMAGE_INDEX, **LABEL**, **OVERLAP**, **X1**,
  Y1, **X2**, **Y2**, **NUM** }

Protected Member Functions

- virtual unsigned int **PrefetchRand** ()
- virtual void **load_batch** (Batch< Dtype > *batch)

Protected Attributes

- shared_ptr< Caffe::RNG > **prefetch_rng_**
- vector< std::pair< std::string, vector< int > > > **image_database_**
- vector< vector< float > > **fg_windows_**
- vector< vector< float > > **bg_windows_**
- Blob< Dtype > **data_mean_**
- vector< Dtype > **mean_values_**
- bool **has_mean_file_**
- bool **has_mean_values_**
- bool **cache_images_**
- vector< std::pair< std::string, Datum > > **image_database_cache_**

5.105.1 Detailed Description

template<typename Dtype>
class caffe::WindowDataLayer< Dtype >

Provides data to the Net from windows of images files, specified by a window data file. This layer is *DEPRECATED* and only kept for archival purposes for use by the original R-CNN.

TODO(dox): thorough documentation for Forward and proto params.

5.105.2 Member Function Documentation

5.105.2.1 ExactNumBottomBlobs()

```
template<typename Dtype >
virtual int caffe::WindowDataLayer< Dtype >::ExactNumBottomBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of bottom blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of bottom blobs.

Reimplemented from caffe::Layer< Dtype >.

5.105.2.2 ExactNumTopBlobs()

```
template<typename Dtype >
virtual int caffe::WindowDataLayer< Dtype >::ExactNumTopBlobs ( ) const  [inline], [virtual]
```

Returns the exact number of top blobs required by the layer, or -1 if no exact number is required.

This method should be overridden to return a non-negative value if your layer expects some exact number of top blobs.

Reimplemented from caffe::Layer< Dtype >.

The documentation for this class was generated from the following file:

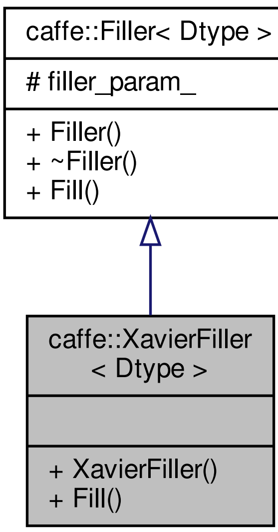
- include/caffe/layers/window_data_layer.hpp

5.106 caffe::XavierFiller< Dtype > Class Template Reference

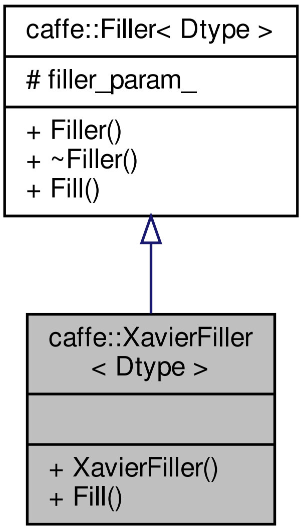
Fills a Blob with values $x \sim U(-a, +a)$ where a is set inversely proportional to number of incoming nodes, outgoing nodes, or their average.

```
#include <filler.hpp>
```

Inheritance diagram for caffe::XavierFiller< Dtype >:



Collaboration diagram for caffe::XavierFiller< Dtype >:



Public Member Functions

- **XavierFiller** (const FillerParameter &param)
- virtual void **Fill** (Blob< Dtype > *blob)

Additional Inherited Members

5.106.1 Detailed Description

template<typename Dtype>
class caffe::XavierFiller< Dtype >

Fills a Blob with values $x \sim U(-a, +a)$ where a is set inversely proportional to number of incoming nodes, outgoing nodes, or their average.

A Filler based on the paper [Bengio and Glorot 2010]: Understanding the difficulty of training deep feedforward neuralnetworks.

It fills the incoming matrix by randomly sampling uniform data from [-scale, scale] where scale = sqrt(3 / n) where n is the fan_in, fan_out, or their average, depending on the variance_norm option. You should make sure the input blob has shape (num, a, b, c) where a $*$ b $*$ c = fan_in and num $*$ b $*$ c = fan_out. Note that this is currently not the case for inner product layers.

TODO(dox): make notation in above comment consistent with rest & use LaTeX.

The documentation for this class was generated from the following file:

- include/caffe/filler.hpp

Index