### Caffe

Generated by Doxygen 1.8.13

# **Contents**

1	Nam	nespace	Index												1
	1.1	Names	space List					 	 1						
2	Hier	archica	l Index												3
	2.1	Class	Hierarchy					 	 	 	 	 	 		 3
3	Clas	s Index													7
	3.1	Class	List					 	 	 	 	 	 		 7
4	Nam	nespace	Documer	ntation											13
	4.1	boost	Namespac	e Referei	nce .			 	 	 	 	 	 		 13
		4.1.1	Detailed	Descripti	on .			 	 	 	 	 	 		 13
	4.2	caffe N	lamespace	e Referen	ce .			 	 	 	 	 	 		 13
		4.2.1	Detailed	Descripti	on .			 	 	 	 	 	 		 27
		4.2.2	Function	Docume	ntation	١		 	 	 	 	 	 		 28
			4.2.2.1	GetFille	r() .			 	 	 	 	 	 		 28
	4.3	caffe::	SolverAction	on Names	pace	Refere	ence	 	 	 	 	 	 		 29
		4.3.1	Detailed	Descripti	on .			 	 	 	 	 	 		 29

ii CONTENTS

5	Clas	s Docu	mentation	1	31
	5.1	caffe::/	AbsValLay	er < Dtype > Class Template Reference	31
		5.1.1	Detailed	Description	34
		5.1.2	Member	Function Documentation	34
			5.1.2.1	Backward_cpu()	34
			5.1.2.2	ExactNumBottomBlobs()	35
			5.1.2.3	ExactNumTopBlobs()	35
			5.1.2.4	Forward_cpu()	35
			5.1.2.5	LayerSetUp()	36
	5.2	caffe::/	AccuracyLa	ayer< Dtype > Class Template Reference	36
		5.2.1	Detailed	Description	39
		5.2.2	Construc	etor & Destructor Documentation	39
			5.2.2.1	AccuracyLayer()	39
		5.2.3	Member	Function Documentation	40
			5.2.3.1	ExactNumBottomBlobs()	40
			5.2.3.2	Forward_cpu()	40
			5.2.3.3	LayerSetUp()	41
			5.2.3.4	MaxTopBlobs()	41
			5.2.3.5	MinTopBlobs()	41
			5.2.3.6	Reshape()	42
	5.3	caffe::/	AdaDeltaS	olver< Dtype > Class Template Reference	43
	5.4	caffe::/	AdaGradSo	olver< Dtype > Class Template Reference	46
	5.5	caffe::/	AdamSolve	er < Dtype > Class Template Reference	48
		5.5.1	Detailed	Description	51
	5.6	caffe::/	ArgMaxLay	yer < Dtype > Class Template Reference	51
		5.6.1	Detailed	Description	54
		5.6.2	Construc	stor & Destructor Documentation	54
			5.6.2.1	ArgMaxLayer()	54
		5.6.3	Member	Function Documentation	55
			5.6.3.1	ExactNumBottomBlobs()	55

CONTENTS

		5.6.3.2	ExactNumTopBlobs()	55
		5.6.3.3	Forward_cpu()	55
		5.6.3.4	LayerSetUp()	56
		5.6.3.5	Reshape()	56
5.7	caffe::E	BaseConvo	olutionLayer < Dtype > Class Template Reference	57
	5.7.1	Detailed	Description	61
	5.7.2	Member	Function Documentation	61
		5.7.2.1	EqualNumBottomTopBlobs()	61
		5.7.2.2	LayerSetUp()	61
		5.7.2.3	MinBottomBlobs()	62
		5.7.2.4	MinTopBlobs()	62
		5.7.2.5	Reshape()	62
5.8	caffe::E	BaseDataL	ayer< Dtype > Class Template Reference	63
	5.8.1	Detailed	Description	65
	5.8.2	Member	Function Documentation	65
		5.8.2.1	LayerSetUp()	65
		5.8.2.2	Reshape()	66
5.9	caffe::E	BasePrefet	chingDataLayer< Dtype > Class Template Reference	67
	5.9.1	Member	Function Documentation	69
		5.9.1.1	LayerSetUp()	69
5.10	caffe::E	Batch < Dty	ype > Class Template Reference	69
5.11	caffe::E	BatchNorm	Layer < Dtype > Class Template Reference	70
	5.11.1	Detailed	Description	73
	5.11.2	Member	Function Documentation	74
		5.11.2.1	ExactNumBottomBlobs()	74
		5.11.2.2	ExactNumTopBlobs()	74
		5.11.2.3	LayerSetUp()	74
		5.11.2.4	Reshape()	75
5.12	caffe::E	BatchReind	dexLayer < Dtype > Class Template Reference	75
	5.12.1	Detailed	Description	78

iv CONTENTS

	5.12.2	Member	Function Documentation	 78
		5.12.2.1	Backward_cpu()	 78
		5.12.2.2	ExactNumBottomBlobs()	 79
		5.12.2.3	ExactNumTopBlobs()	 79
		5.12.2.4	Forward_cpu()	 79
		5.12.2.5	Reshape()	 80
5.13	caffe::E	BiasLayer<	< Dtype > Class Template Reference	 80
	5.13.1	Detailed	Description	 83
	5.13.2	Member	Function Documentation	 83
		5.13.2.1	ExactNumTopBlobs()	 83
		5.13.2.2	LayerSetUp()	 84
		5.13.2.3	MaxBottomBlobs()	 85
		5.13.2.4	MinBottomBlobs()	 85
		5.13.2.5	Reshape()	 85
5.14	caffe::E	BilinearFille	er < Dtype > Class Template Reference	 86
	5.14.1	Detailed	Description	 87
5.15	caffe::E	Blob< Dtyp	pe > Class Template Reference	 88
	5.15.1	Detailed	Description	 91
	5.15.2	Member	Function Documentation	 91
		5.15.2.1	CanonicalAxisIndex()	 91
		5.15.2.2	CopyFrom()	 91
		5.15.2.3	count() [1/2]	 92
		5.15.2.4	count() [2/2]	 92
		5.15.2.5	Reshape()	 92
		5.15.2.6	shape()	 93
		5.15.2.7	ShareData()	 93
		5.15.2.8	ShareDiff()	 93
5.16	caffe::E	BlockingQu	ueue < T > Class Template Reference	 94
5.17	caffe::E	BNLLLayer	r < Dtype > Class Template Reference	 95
	5.17.1	Detailed	Description	 98

CONTENTS

	5.17.2	Member I	Function Documentation	98
		5.17.2.1	Backward_cpu()	98
		5.17.2.2	Forward_cpu()	99
5.18	caffe::C	Caffe Class	s Reference	100
5.19	caffe::N	let< Dtype	e >::Callback Class Reference	101
5.20	caffe::S	Solver< Dt	ype >::Callback Class Reference	102
5.21	caffe::C	ClipLayer<	Dtype > Class Template Reference	103
	5.21.1	Detailed I	Description	105
	5.21.2	Construc	tor & Destructor Documentation	105
		5.21.2.1	ClipLayer()	105
	5.21.3	Member I	Function Documentation	105
		5.21.3.1	Backward_cpu()	106
		5.21.3.2	Forward_cpu()	106
5.22	caffe::C	ConcatLaye	er < Dtype > Class Template Reference	107
	5.22.1	Detailed I	Description	109
	5.22.2	Member I	Function Documentation	109
		5.22.2.1	Backward_cpu()	109
		5.22.2.2	ExactNumTopBlobs()	110
		5.22.2.3	Forward_cpu()	110
		5.22.2.4	LayerSetUp()	111
		5.22.2.5	MinBottomBlobs()	111
		5.22.2.6	Reshape()	111
5.23	caffe::C	ConstantFil	Iller< Dtype > Class Template Reference	112
	5.23.1	Detailed I	Description	113
5.24	caffe::C	Contrastive	LossLayer < Dtype > Class Template Reference	114
	5.24.1	Detailed I	Description	116
	5.24.2	Member I	Function Documentation	117
		5.24.2.1	AllowForceBackward()	117
		5.24.2.2	Backward_cpu()	117
		5.24.2.3	ExactNumBottomBlobs()	118

vi

5.24.2.4 Forward_cpu()	118
5.24.2.5 LayerSetUp()	118
5.25 caffe::ConvolutionLayer< Dtype > Class Template Reference	119
5.25.1 Detailed Description	122
5.25.2 Constructor & Destructor Documentation	122
5.25.2.1 ConvolutionLayer()	122
5.26 caffe::CPUTimer Class Reference	124
5.27 caffe::CropLayer < Dtype > Class Template Reference	126
5.27.1 Detailed Description	128
5.27.2 Member Function Documentation	128
5.27.2.1 ExactNumBottomBlobs()	128
5.27.2.2 ExactNumTopBlobs()	129
5.27.2.3 LayerSetUp()	129
5.27.2.4 Reshape()	129
5.28 caffe::db::Cursor Class Reference	130
5.29 caffe::DataLayer < Dtype > Class Template Reference	131
5.29.1 Member Function Documentation	133
5.29.1.1 ExactNumBottomBlobs()	133
5.29.1.2 MaxTopBlobs()	133
5.29.1.3 MinTopBlobs()	133
5.30 caffe::DataTransformer< Dtype > Class Template Reference	134
5.30.1 Detailed Description	135
5.30.2 Member Function Documentation	135
5.30.2.1 InferBlobShape() [1/2]	135
<b>5.30.2.2</b> InferBlobShape() [2/2]	136
5.30.2.3 Rand()	136
<b>5.30.2.4 Transform()</b> [1/3]	136
<b>5.30.2.5 Transform()</b> [2/3]	137
<b>5.30.2.6 Transform()</b> [3/3]	137
5.31 caffe::db::DB Class Reference	138

CONTENTS vii

5.32	caffe::E	DeconvolutionLayer< Dtype > Class Template Reference	38
	5.32.1	Detailed Description	41
5.33	caffe::E	DropoutLayer < Dtype > Class Template Reference	41
	5.33.1	Detailed Description	44
	5.33.2	Constructor & Destructor Documentation	44
		5.33.2.1 DropoutLayer()	45
	5.33.3	Member Function Documentation	45
		5.33.3.1 Forward_cpu()	45
		5.33.3.2 LayerSetUp()	45
		5.33.3.3 Reshape()	46
5.34	caffe::E	DummyDataLayer < Dtype > Class Template Reference	46
	5.34.1	Detailed Description	49
	5.34.2	Member Function Documentation	49
		5.34.2.1 ExactNumBottomBlobs()	49
		5.34.2.2 LayerSetUp()	49
		5.34.2.3 MinTopBlobs()	50
		5.34.2.4 Reshape()	50
5.35	caffe::E	EltwiseLayer < Dtype > Class Template Reference	51
	5.35.1	Detailed Description	53
	5.35.2	Member Function Documentation	53
		5.35.2.1 ExactNumTopBlobs()	53
		5.35.2.2 LayerSetUp()	53
		5.35.2.3 MinBottomBlobs()	54
		5.35.2.4 Reshape()	54
5.36	caffe::E	ELULayer < Dtype > Class Template Reference	55
	5.36.1	Detailed Description	57
	5.36.2	Constructor & Destructor Documentation	57
		5.36.2.1 ELULayer()	57
	5.36.3	Member Function Documentation	57
		5.36.3.1 Backward_cpu()	58

viii CONTENTS

		5.36.3.2 Forward_cpu()		 	 	158
5.37	caffe::E	mbedLayer< Dtype > Class Template	e Reference	 	 	159
	5.37.1	Detailed Description		 	 	161
	5.37.2	Member Function Documentation .		 	 	161
		5.37.2.1 ExactNumBottomBlobs() .		 	 	162
		5.37.2.2 ExactNumTopBlobs()		 	 	162
		5.37.2.3 LayerSetUp()		 	 	162
		5.37.2.4 Reshape()		 	 	163
5.38	caffe::E	uclideanLossLayer< Dtype > Class T	emplate Reference	 	 	163
	5.38.1	Detailed Description		 	 	166
	5.38.2	Member Function Documentation .		 	 	167
		5.38.2.1 AllowForceBackward()		 	 	167
		5.38.2.2 Backward_cpu()		 	 	167
		5.38.2.3 Forward_cpu()		 	 	168
		5.38.2.4 Reshape()		 	 	168
5.39	caffe::E	xpLayer< Dtype > Class Template Re	eference	 	 	169
	5.39.1	Detailed Description		 	 	172
	5.39.2	Constructor & Destructor Documenta	tion	 	 	172
		5.39.2.1 ExpLayer()		 	 	172
	5.39.3	Member Function Documentation .		 	 	173
		5.39.3.1 Backward_cpu()		 	 	173
		5.39.3.2 Forward_cpu()		 	 	173
		5.39.3.3 LayerSetUp()		 	 	174
5.40	caffe::F	iller< Dtype > Class Template Refere	nce	 	 	174
	5.40.1	Detailed Description		 	 	175
5.41	caffe::F	ilterLayer< Dtype > Class Template F	Reference	 	 	175
	5.41.1	Detailed Description		 	 	178
	5.41.2	Member Function Documentation .		 	 	178
		5.41.2.1 Backward_cpu()		 	 	178
		5.41.2.2 Forward_cpu()		 	 	179

CONTENTS

		5.41.2.3 I	LayerSetUp()	. 179
		5.41.2.4	MinBottomBlobs()	. 180
		5.41.2.5 I	MinTopBlobs()	. 180
		5.41.2.6 I	Reshape()	. 180
5.42	caffe::F	- lattenLayer	< Dtype > Class Template Reference	. 181
	5.42.1	Detailed D	escription	. 184
	5.42.2	Member Fi	unction Documentation	. 184
		5.42.2.1 I	Backward_cpu()	. 184
		5.42.2.2	ExactNumBottomBlobs()	. 185
		5.42.2.3 I	ExactNumTopBlobs()	. 185
		5.42.2.4	Forward_cpu()	. 185
		5.42.2.5 I	Reshape()	. 185
5.43	caffe::G	SaussianFill	er < Dtype > Class Template Reference	. 186
	5.43.1	Detailed D	escription	. 187
5.44	caffe::C	Caffe::RNG::	:Generator Class Reference	. 188
5.45	caffe::H	IDF5DataLa	ayer < Dtype > Class Template Reference	. 188
	5.45.1	Detailed D	escription	. 191
	5.45.2	Member Fi	unction Documentation	. 191
		5.45.2.1 I	ExactNumBottomBlobs()	. 192
		5.45.2.2 I	LayerSetUp()	. 192
		5.45.2.3 I	MinTopBlobs()	. 192
		5.45.2.4	Reshape()	. 193
5.46	caffe::F	IDF5Output	tLayer < Dtype > Class Template Reference	. 194
	5.46.1	Detailed D	escription	. 197
	5.46.2	Member Fi	unction Documentation	. 197
		5.46.2.1 I	ExactNumBottomBlobs()	. 198
		5.46.2.2	ExactNumTopBlobs()	. 198
		5.46.2.3 I	LayerSetUp()	. 198
		5.46.2.4	Reshape()	. 199
5.47	caffe::F	lingeLossLa	ayer < Dtype > Class Template Reference	. 199

CONTENTS

	5.47.1	Detailed I	Description	. 202
	5.47.2	Member F	Function Documentation	. 202
		5.47.2.1	Backward_cpu()	. 203
		5.47.2.2	Forward_cpu()	. 203
5.48	caffe::li	m2colLaye	er < Dtype > Class Template Reference	. 204
	5.48.1	Detailed I	Description	. 207
	5.48.2	Member F	Function Documentation	. 207
		5.48.2.1	ExactNumBottomBlobs()	. 208
		5.48.2.2	ExactNumTopBlobs()	. 208
		5.48.2.3	LayerSetUp()	. 208
		5.48.2.4	Reshape()	. 209
5.49	caffe::li	mageDatal	Layer < Dtype > Class Template Reference	. 209
	5.49.1	Detailed I	Description	. 212
	5.49.2	Member F	Function Documentation	. 212
		5.49.2.1	ExactNumBottomBlobs()	. 212
		5.49.2.2	ExactNumTopBlobs()	. 212
5.50	caffe::li	nfogainLos	ssLayer< Dtype > Class Template Reference	. 213
	5.50.1	Detailed I	Description	. 216
	5.50.2	Member F	Function Documentation	. 216
		5.50.2.1	Backward_cpu()	. 216
		5.50.2.2	ExactNumBottomBlobs()	. 217
		5.50.2.3	ExactNumTopBlobs()	. 217
		5.50.2.4	Forward_cpu()	. 218
		5.50.2.5	get_normalizer()	. 219
		5.50.2.6	LayerSetUp()	. 219
		5.50.2.7	MaxBottomBlobs()	. 220
		5.50.2.8	MaxTopBlobs()	. 220
		5.50.2.9	MinBottomBlobs()	. 220
		5.50.2.10	MinTopBlobs()	. 221
		5.50.2.11	Reshape()	. 221

CONTENTS xi

5.51	caffe::Ir	nnerProdu	tLayer< Dtype > 0	Class Templa	te Refere	ence .	 	 	 	221
	5.51.1	Detailed	escription				 	 	 	224
	5.51.2	Member	unction Document	ation			 	 	 	224
		5.51.2.1	ExactNumBottomE	Blobs()			 	 	 	225
		5.51.2.2	ExactNumTopBlob	s()			 	 	 	225
		5.51.2.3	LayerSetUp()				 	 	 	225
		5.51.2.4	Reshape()				 	 	 	226
5.52	caffe::Ir	nputLayer	Collapse Dtype > Class Te	mplate Refer	ence .		 	 	 	226
	5.52.1	Detailed	Description				 	 	 	229
	5.52.2	Member	unction Document	ation			 	 	 	229
		5.52.2.1	ExactNumBottomE	Blobs()			 	 	 	229
		5.52.2.2	LayerSetUp()				 	 	 	229
		5.52.2.3	MinTopBlobs()				 	 	 	230
		5.52.2.4	Reshape()				 	 	 	230
5.53	caffe::Ir	nternalThr	ead Class Reference	e			 	 	 	231
	5.53.1	Detailed	Description				 	 	 	232
	5.53.2	Member	unction Document	ation			 	 	 	232
		5.53.2.1	StartInternalThrea	d()			 	 	 	232
		5.53.2.2	StopInternalThrea	d()			 	 	 	233
5.54	caffe::L	.ayer< Dty	pe > Class Templa	te Reference			 	 	 	233
	5.54.1	Detailed	escription				 	 	 	235
	5.54.2	Construc	or & Destructor Do	cumentation .			 	 	 	235
		5.54.2.1	Layer()				 	 	 	235
	5.54.3	Member	unction Document	ation			 	 	 	236
		5.54.3.1	AllowForceBackwa	ard()			 	 	 	236
		5.54.3.2	AutoTopBlobs()				 	 	 	236
		5.54.3.3	Backward()				 	 	 	236
		5.54.3.4	CheckBlobCounts	)			 	 	 	237
		5.54.3.5	EqualNumBottom	TopBlobs()			 	 	 	237
		5.54.3.6	ExactNumBottomE	Blobs()			 	 	 	237

xii CONTENTS

		5.54.3.7	ExactNumTopBlobs()	238
		5.54.3.8	Forward()	238
		5.54.3.9	LayerSetUp()	238
		5.54.3.10	MaxBottomBlobs()	239
		5.54.3.11	MaxTopBlobs()	239
		5.54.3.12	MinBottomBlobs()	240
		5.54.3.13	MinTopBlobs()	240
		5.54.3.14	param_propagate_down()	240
		5.54.3.15	Reshape()	240
		5.54.3.16	SetLossWeights()	241
		5.54.3.17	SetUp()	241
	5.54.4	Member [	Data Documentation	242
		5.54.4.1	blobs	242
		5.54.4.2	layer_param	242
		5.54.4.3	loss	242
		5.54.4.4	param_propagate_down	242
		5.54.4.5	phase	242
5.55	caffe::L	.ayerRegist	terer< Dtype > Class Template Reference	243
5.56	caffe::L	.ayerRegis	try< Dtype > Class Template Reference	243
5.57	caffe::L	.ogLayer<	Dtype > Class Template Reference	244
	5.57.1	Detailed [	Description	247
	5.57.2	Construct	tor & Destructor Documentation	247
		5.57.2.1	LogLayer()	247
	5.57.3	Member F	Function Documentation	248
		5.57.3.1	Backward_cpu()	248
		5.57.3.2	Forward_cpu()	248
		5.57.3.3	LayerSetUp()	249
5.58	caffe::L	.ossLayer<	Color > Class Template Reference	249
	5.58.1	Detailed [	Description	251
	5.58.2	Member F	Function Documentation	251

CONTENTS xiii

		5.58.2.1	AllowForceBackward()	251
		5.58.2.2	ExactNumBottomBlobs()	251
		5.58.2.3	ExactNumTopBlobs()	252
		5.58.2.4	LayerSetUp()	252
		5.58.2.5	Reshape()	252
5.59	caffe::L	.RNLayer<	Collaboration Characteristics Characteristics Collaboration Collaboratio	253
	5.59.1	Detailed	Description	257
	5.59.2	Member	Function Documentation	257
		5.59.2.1	ExactNumBottomBlobs()	257
		5.59.2.2	ExactNumTopBlobs()	257
		5.59.2.3	LayerSetUp()	257
		5.59.2.4	Reshape()	258
5.60	caffe::L	.STMLaye	r< Dtype > Class Template Reference	258
	5.60.1	Detailed	Description	261
5.61	caffe::L	.STMUnitL	ayer< Dtype > Class Template Reference	262
	5.61.1	Detailed	Description	264
	5.61.2	Member	Function Documentation	264
		5.61.2.1	AllowForceBackward()	264
		5.61.2.2	Backward_cpu()	265
		5.61.2.3	ExactNumBottomBlobs()	265
		5.61.2.4	ExactNumTopBlobs()	265
		5.61.2.5	Forward_cpu()	266
		5.61.2.6	Reshape()	266
5.62	caffe::N	/lemoryDa	taLayer< Dtype > Class Template Reference	267
	5.62.1	Detailed	Description	269
	5.62.2	Member	Function Documentation	269
		5.62.2.1	ExactNumBottomBlobs()	270
		5.62.2.2	ExactNumTopBlobs()	270
5.63	caffe::N	//SRAFiller	<pre>r&lt; Dtype &gt; Class Template Reference</pre>	270
	5.63.1	Detailed	Description	272

xiv CONTENTS

5.64	caffe::N	fultinomialLogisticLossLayer         Dtype > Class Template Reference	72
	5.64.1	Detailed Description	75
	5.64.2	Member Function Documentation	75
		5.64.2.1 Backward_cpu()	75
		5.64.2.2 Forward_cpu()	76
		5.64.2.3 Reshape()	77
5.65	caffe::N	IVNLayer < Dtype > Class Template Reference	77
	5.65.1	Detailed Description	30
	5.65.2	Member Function Documentation	30
		5.65.2.1 ExactNumBottomBlobs()	31
		5.65.2.2 ExactNumTopBlobs()	31
		5.65.2.3 Reshape()	31
5.66	caffe::N	lesterovSolver< Dtype > Class Template Reference	32
5.67	caffe::N	let < Dtype > Class Template Reference	34
	5.67.1	Detailed Description	38
	5.67.2	Member Function Documentation	38
		5.67.2.1 Backward()	38
		5.67.2.2 ForwardFromTo()	38
		5.67.2.3 Reshape()	38
		5.67.2.4 ShareWeights()	38
	5.67.3	Member Data Documentation	39
		5.67.3.1 blob_loss_weights	39
		5.67.3.2 bottom_vecs	39
		5.67.3.3   learnable_param_ids	39
5.68	caffe::N	leuronLayer< Dtype > Class Template Reference	39
	5.68.1	Detailed Description	91
	5.68.2	Member Function Documentation	91
		5.68.2.1 ExactNumBottomBlobs()	91
		5.68.2.2 ExactNumTopBlobs()	91
		5.68.2.3 Reshape()	91

CONTENTS xv

5.69 caffe::	ParameterLayer < Dtype > Class Template Reference
5.69.1	Member Function Documentation
	5.69.1.1 ExactNumBottomBlobs()
	5.69.1.2 ExactNumTopBlobs()
	5.69.1.3 LayerSetUp()
	5.69.1.4 Reshape()
5.70 caffe::	PoolingLayer < Dtype > Class Template Reference
5.70.1	Detailed Description
5.70.2	Member Function Documentation
	5.70.2.1 ExactNumBottomBlobs()
	5.70.2.2 LayerSetUp()
	5.70.2.3 MaxTopBlobs()
	5.70.2.4 MinTopBlobs()
	5.70.2.5 Reshape()
5.71 caffe::	PositiveUnitballFiller< Dtype > Class Template Reference
5.71.1	Detailed Description
5.72 caffe::	PowerLayer < Dtype > Class Template Reference
5.72.1	Detailed Description
5.72.2	Constructor & Destructor Documentation
	5.72.2.1 PowerLayer()
5.72.3	Member Function Documentation
	5.72.3.1 Backward_cpu()
	5.72.3.2 Forward_cpu()
	5.72.3.3 LayerSetUp()
5.73 caffe::	PReLULayer < Dtype > Class Template Reference
5.73.1	Detailed Description
5.73.2	2 Constructor & Destructor Documentation
	5.73.2.1 PReLULayer()
5.73.3	Member Function Documentation
	5.73.3.1 Backward_cpu()

xvi CONTENTS

		5.73.3.2	Forward_cpu()	312
		5.73.3.3	LayerSetUp()	313
		5.73.3.4	Reshape()	313
5.74	caffe::F	ythonLaye	er < Dtype > Class Template Reference	315
	5.74.1	Member I	Function Documentation	317
		5.74.1.1	LayerSetUp()	317
		5.74.1.2	Reshape()	317
5.75	caffe::F	RecurrentL	.ayer< Dtype > Class Template Reference	318
	5.75.1	Detailed I	Description	322
	5.75.2	Member I	Function Documentation	322
		5.75.2.1	AllowForceBackward()	322
		5.75.2.2	ExactNumTopBlobs()	322
		5.75.2.3	Forward_cpu()	322
		5.75.2.4	LayerSetUp()	323
		5.75.2.5	MaxBottomBlobs()	324
		5.75.2.6	MinBottomBlobs()	324
		5.75.2.7	Reshape()	324
5.76	caffe::F	ReductionL	_ayer< Dtype > Class Template Reference	325
	5.76.1	Detailed I	Description	328
	5.76.2	Member I	Function Documentation	329
		5.76.2.1	ExactNumBottomBlobs()	329
		5.76.2.2	ExactNumTopBlobs()	329
		5.76.2.3	LayerSetUp()	329
		5.76.2.4	Reshape()	330
5.77	caffe::F	ReLULayer	r < Dtype > Class Template Reference	330
	5.77.1	Detailed I	Description	333
	5.77.2	Construc	etor & Destructor Documentation	333
		5.77.2.1	ReLULayer()	333
	5.77.3	Member I	Function Documentation	333
		5.77.3.1	Backward_cpu()	334

CONTENTS xvii

		5.77.3.2	Forward_cpu()	 334
5.78	caffe::F	ReshapeLa	ayer < Dtype > Class Template Reference	 335
	5.78.1	Member I	Function Documentation	 337
		5.78.1.1	ExactNumBottomBlobs()	 337
		5.78.1.2	ExactNumTopBlobs()	 338
		5.78.1.3	LayerSetUp()	 338
		5.78.1.4	Reshape()	 338
5.79	caffe::F	RMSPropS	Solver< Dtype > Class Template Reference	 340
5.80	caffe::C	Caffe::RNG	G Class Reference	 342
5.81	caffe::F	RNNLayer<	< Dtype > Class Template Reference	 343
	5.81.1	Detailed I	Description	 345
5.82	caffe::S	ScaleLayer	r < Dtype > Class Template Reference	 345
	5.82.1	Detailed I	Description	 348
	5.82.2	Member I	Function Documentation	 348
		5.82.2.1	ExactNumTopBlobs()	 348
		5.82.2.2	Forward_cpu()	 349
		5.82.2.3	LayerSetUp()	 349
		5.82.2.4	MaxBottomBlobs()	 349
		5.82.2.5	MinBottomBlobs()	 350
		5.82.2.6	Reshape()	 350
5.83	caffe::S	GDSolver	r< Dtype > Class Template Reference	 351
	5.83.1	Detailed I	Description	 353
5.84	caffe::S	SigmoidCro	ossEntropyLossLayer< Dtype > Class Template Reference	 353
	5.84.1	Detailed I	Description	 356
	5.84.2	Member I	Function Documentation	 357
			Backward_cpu()	
			Forward_cpu()	
			get_normalizer()	
			LayerSetUp()	
			Reshape()	
		5.57.2.5		

xviii CONTENTS

5.85	caffe::S	SigmoidLayer< Dtype > Class Template Reference	59
	5.85.1	Detailed Description	62
	5.85.2	Member Function Documentation	62
		5.85.2.1 Backward_cpu()	62
		5.85.2.2 Forward_cpu()	63
5.86	caffe::S	SignalHandler Class Reference	63
5.87	caffe::S	SilenceLayer < Dtype > Class Template Reference	64
	5.87.1	Detailed Description	66
	5.87.2	Member Function Documentation	66
		5.87.2.1 ExactNumTopBlobs()	66
		5.87.2.2 MinBottomBlobs()	67
		5.87.2.3 Reshape()	67
5.88	caffe::S	SliceLayer < Dtype > Class Template Reference	67
	5.88.1	Detailed Description	70
	5.88.2	Member Function Documentation	70
		5.88.2.1 ExactNumBottomBlobs()	71
		5.88.2.2 LayerSetUp()	71
		5.88.2.3 MinTopBlobs()	71
		5.88.2.4 Reshape()	72
5.89	caffe::S	SoftmaxLayer < Dtype > Class Template Reference	73
	5.89.1	Detailed Description	76
	5.89.2	Member Function Documentation	76
		5.89.2.1 ExactNumBottomBlobs()	77
		5.89.2.2 ExactNumTopBlobs()	77
		5.89.2.3 Reshape()	77
5.90	caffe::S	SoftmaxWithLossLayer < Dtype > Class Template Reference	78
	5.90.1	Detailed Description	81
	5.90.2	Constructor & Destructor Documentation	82
		5.90.2.1 SoftmaxWithLossLayer()	82
	5.90.3	Member Function Documentation	82

CONTENTS xix

		5.90.3.1	Backward_cpu()	 382
		5.90.3.2	ExactNumTopBlobs()	 383
		5.90.3.3	get_normalizer()	 383
		5.90.3.4	LayerSetUp()	 383
		5.90.3.5	MaxTopBlobs()	 384
		5.90.3.6	MinTopBlobs()	 384
		5.90.3.7	Reshape()	 384
5.91	caffe::S	Solver< Dt	type > Class Template Reference	 385
	5.91.1	Detailed	Description	 387
5.92	caffe::S	SolverRegi	isterer< Dtype > Class Template Reference	 388
5.93	caffe::S	SolverRegi	istry < Dtype > Class Template Reference	 388
5.94	caffe::S	SplitLayer<	< Dtype > Class Template Reference	 389
	5.94.1	Detailed	Description	 392
	5.94.2	Member	Function Documentation	 392
		5.94.2.1	ExactNumBottomBlobs()	 392
		5.94.2.2	MinTopBlobs()	 393
		5.94.2.3	Reshape()	 393
5.95	caffe::S	SPPLayer<	< Dtype > Class Template Reference	 393
	5.95.1	Detailed	Description	 396
	5.95.2	Member	Function Documentation	 396
		5.95.2.1	ExactNumBottomBlobs()	 397
		5.95.2.2	ExactNumTopBlobs()	 397
		5.95.2.3	LayerSetUp()	 397
		5.95.2.4	Reshape()	 398
5.96	caffe::S	SwishLaye	r< Dtype > Class Template Reference	 398
	5.96.1	Detailed	Description	 401
	5.96.2	Construc	etor & Destructor Documentation	 401
		5.96.2.1	SwishLayer()	 401
	5.96.3	Member	Function Documentation	 402
		5.96.3.1	Backward_cpu()	 402

XX CONTENTS

Ind	dex	429
	5.106.1 Detailed Description	427
	5.106caffe::XavierFiller < Dtype > Class Template Reference	426
	5.105.2.2 ExactNumTopBlobs()	426
	5.105.2.1 ExactNumBottomBlobs()	425
	5.105.2 Member Function Documentation	425
	5.105.1 Detailed Description	425
	5.105caffe::WindowDataLayer< Dtype > Class Template Reference	422
	5.104.1 Detailed Description	422
	5.104caffe::UniformFiller < Dtype > Class Template Reference	421
	5.103caffe::db::Transaction Class Reference	420
	5.102caffe::Timer Class Reference	418
	5.101.2.3 Reshape()	417
	5.101.2.2 ExactNumTopBlobs()	417
	5.101.2.1 ExactNumBottomBlobs()	416
	5.101.2 Member Function Documentation	416
	5.101.1 Detailed Description	416
	5.101 caffe::TileLayer< Dtype > Class Template Reference	413
	5.100.3.2 LayerSetUp()	413
	5.100.3.1 Forward_cpu()	412
	5.100.3 Member Function Documentation	412
	5.100.2.1 ThresholdLayer()	412
	5.100.2 Constructor & Destructor Documentation	412
	5.100.1 Detailed Description	412
	5.100caffe::ThresholdLayer< Dtype > Class Template Reference	409
	5.99.2.2 Forward_cpu()	409
	5.99.2.1 Backward_cpu()	408
	5.99.2 Member Function Documentation	408
	5.99.1 Detailed Description	408
	5.99 caffe::TanHLayer< Dtype > Class Template Reference	405
	5.98.1 Detailed Description	405
	5.98 caffe::SyncedMemory Class Reference	404
	5.97 caffe::BlockingQueue < T >::sync Class Reference	404
	5.96.3.4 Reshape()	403
	5.96.3.3 LayerSetUp()	403
	5.96.3.2 Forward_cpu()	402

### **Chapter 1**

# Namespace Index

#### 1.1 Namespace List

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

boost . caffe		13
	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:	13
caffe::So	olverAction	
	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	
	snanshot to be saved when stopping execution with a SIGINT (Ctrl-C)	20

2 Namespace Index

# Chapter 2

# **Hierarchical Index**

### 2.1 Class Hierarchy

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

caffe::Batch< Dtype >
caffe::Blob< Dtype >
$caffe :: Blob < int > \dots $
caffe::Blob< unsigned int >
$caffe:: Blocking Queue < T > \dots \dots$
$caffe:: Blocking Queue < caffe:: Batch < Dtype > *> \dots $
caffe::Caffe
caffe::Net< Dtype >::Callback
caffe::Solver< Dtype >::Callback
caffe::db::Cursor
caffe::DataTransformer< Dtype >
caffe::db::DB
caffe::Filler < Dtype >
caffe::BilinearFiller< Dtype >
caffe::ConstantFiller < Dtype >
caffe::GaussianFiller < Dtype >
caffe::MSRAFiller< Dtype >
caffe::PositiveUnitballFiller< Dtype >
caffe::UniformFiller< Dtype >
caffe::XavierFiller < Dtype >
caffe::Caffe::RNG::Generator
caffe::InternalThread
caffe::BasePrefetchingDataLayer< Dtype >
caffe::DataLayer< Dtype >
caffe::ImageDataLayer< Dtype >
caffe::WindowDataLayer< Dtype >
caffe::Layer< Dtype >
caffe::AccuracyLayer< Dtype >
caffe::ArgMaxLayer< Dtype >
caffe::BaseConvolutionLayer< Dtype >
caffe::ConvolutionLayer< Dtype >
caffe::DeconvolutionLayer< Dtype >
caffe::BaseDataLayer< Dtype >
caffe::BasePrefetchingDataLayer< Dtype >

4 Hierarchical Index

${\sf caffe::} {\sf MemoryDataLayer} {\sf < Dtype} {\sf > $	. 267
caffe::BatchNormLayer< Dtype >	. 70
caffe::BatchReindexLayer< Dtype >	. 75
caffe::BiasLayer< Dtype >	. 80
caffe::ConcatLayer< Dtype >	. 107
caffe::CropLayer< Dtype >	. 126
caffe::DummyDataLayer< Dtype >	
caffe::EltwiseLayer< Dtype >	
caffe::EmbedLayer< Dtype >	. 159
caffe::FilterLayer< Dtype >	. 175
caffe::FlattenLayer < Dtype >	. 181
caffe::HDF5DataLayer< Dtype >	. 188
caffe::HDF5OutputLayer< Dtype >	. 194
caffe::Im2colLayer< Dtype >	. 204
caffe::InnerProductLayer< Dtype >	. 221
caffe::InputLayer< Dtype >	. 226
caffe::LossLayer< Dtype >	. 249
caffe::ContrastiveLossLayer < Dtype >	. 114
caffe::EuclideanLossLayer< Dtype >	
caffe::HingeLossLayer < Dtype >	. 199
caffe::InfogainLossLayer< Dtype >	
caffe::MultinomialLogisticLossLayer< Dtype >	
caffe::SigmoidCrossEntropyLossLayer< Dtype >	
caffe::SoftmaxWithLossLayer< Dtype >	
caffe::LRNLayer< Dtype >	
caffe::LSTMUnitLayer< Dtype >	
caffe::MVNLayer< Dtype >	
caffe::NeuronLayer< Dtype >	
caffe::AbsValLayer< Dtype >	
caffe::BNLLLayer< Dtype >	
caffe::ClipLayer< Dtype >	
caffe::DropoutLayer< Dtype >	
caffe::ELULayer< Dtype >	
caffe::ExpLayer< Dtype >	
caffe::LogLayer< Dtype >	
caffe::PowerLayer< Dtype >	
caffe::PReLULayer< Dtype >	
caffe::ReLULayer< Dtype >	
caffe::SigmoidLayer< Dtype >	
caffe::SwishLayer< Dtype >	
caffe::TanHLayer< Dtype >	
caffe::ThresholdLayer< Dtype >	
caffe::ParameterLayer < Dtype >	
caffe::PoolingLayer< Dtype >	
caffe::PythonLayer< Dtype >	
caffe::RecurrentLayer< Dtype >	
caffe::LSTMLayer< Dtype >	
caffe::RNNLayer< Dtype >	
caffe::ReductionLayer< Dtype >	
caffe::ReshapeLayer < Dtype >	
caffe::ScaleLayer < Dtype >	
caffe::SilenceLayer < Dtype >	
caffe::SliceLayer< Dtype >	
caffe::SoftmaxLayer< Dtype >	
caffe::SplitLayer< Dtype >	
caffe::SPPLayer < Dtype >	
caffe::TileLayer < Dtype >	
caffe::LayerRegisterer< Dtype >	
caneLayen registerer < Dtype /	243

2.1 Class Hierarchy 5

caffe::LayerRegistry < Dtype >
caffe::Net< Dtype >
caffe::Caffe::RNG
caffe::SignalHandler
caffe::Solver< Dtype >
caffe::SGDSolver< Dtype >
caffe::AdaDeltaSolver< Dtype >
caffe::AdaGradSolver< Dtype >
caffe::AdamSolver< Dtype >
caffe::NesterovSolver< Dtype >
caffe::RMSPropSolver< Dtype >
caffe::SolverRegisterer< Dtype >
${\sf caffe::} {\sf SolverRegistry} {\sf < Dtype} {\sf > $
$caffe:: Blocking Queue < T > :: sync \dots 404000000000000000000000000000000000$
caffe::SyncedMemory
caffe::Timer
caffe::CPUTimer
caffe::db::Transaction

6 Hierarchical Index

# **Chapter 3**

# **Class Index**

#### 3.1 Class List

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

Computes the classification accuracy for a one-of-many classification task  caffe::AdaDeltaSolver
caffe::AdaDeltaSolver
$ \begin{array}{llllllllllllllllllllllllllllllllllll$
caffe::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]
tions, based on adaptive estimates of lower-order moments. Described in [1]
$ \begin{array}{llllllllllllllllllllllllllllllllllll$
Compute the index of the $K$ max values for each datum across all dimensions $(C \times H \times W)$ 5 caffe::BaseConvolutionLayer < Dtype > Abstract base class that factors out the BLAS code common to ConvolutionLayer and DeconvolutionLayer
caffe::BaseConvolutionLayer < Dtype > Abstract base class that factors out the BLAS code common to ConvolutionLayer and DeconvolutionLayer
Abstract base class that factors out the BLAS code common to ConvolutionLayer and DeconvolutionLayer
DeconvolutionLayer
caffe::BaseDataLayer< Dtype > Provides base for data layers that feed blobs to the Net 65 caffe::BasePrefetchingDataLayer< Dtype > 65 caffe::Batch< Dtype > 65 caffe::BatchNormLayer< Dtype > Normalizes the input to have 0-mean and/or unit (1) variance across the batch 76 caffe::BatchReindexLayer< Dtype > Index into the input blob along its first axis 75 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 66 caffe::BilinearFiller< Dtype > Fills a Blob with coefficients for bilinear interpolation 86 caffe::Blob< Dtype >
Provides base for data layers that feed blobs to the Net
caffe::BasePrefetchingDataLayer < Dtype >
caffe::Batch< Dtype >
caffe::BatchNormLayer< Dtype >     Normalizes the input to have 0-mean and/or unit (1) variance across the batch
Normalizes the input to have 0-mean and/or unit (1) variance across the batch
caffe::BatchReindexLayer< Dtype >
Index into the input blob along its first axis
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 . 80 caffe::BilinearFiller< Dtype > Fills a Blob with coefficients for bilinear interpolation
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 . 80 caffe::BilinearFiller< Dtype > Fills a Blob with coefficients for bilinear interpolation
shape of the former. Equivalent to tiling the latter Blob, then computing the elementwise sum caffe::BilinearFiller< Dtype > Fills a Blob with coefficients for bilinear interpolation
caffe::BilinearFiller< Dtype > Fills a Blob with coefficients for bilinear interpolation
Fills a Blob with coefficients for bilinear interpolation
caffe::Blob < Dtype >
A wrapper around Syncedimemory holders serving as the basic computational unit through which
Layers, Nets, and Solvers interact
caffe::BlockingQueue< T >
caffe::BNLLLayer< Dtype > Computes $y = x + \log(1 + \exp(-x))$ if $x > 0$ ; $y = \log(1 + \exp(x))$ otherwise
caffe::Caffe

8 Class Index

caffe::Solver< Dtype >::Callback	102
$\label{eq:caffe::ClipLayer} \begin{aligned} Clip: \ y &= \max(min, \min(max, x)) \\ & \dots \\ \\ & \dots \\ & \dots \\ \\ & \dots \\ & \dots \\ \\ \\ & \dots \\ \\ & \dots \\ \\ \\ & \dots \\ \\ & \dots \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ $	103
caffe::ConcatLayer< Dtype >	
Takes at least two Blobs and concatenates them along either the num or channel dimension,	40-
outputting the result	107
Fills a Blob with constant values $x=0$	112
caffe::ContrastiveLossLayer< Dtype >	
Computes the contrastive loss $E=\frac{1}{2N}\sum_{n=1}^{N}\left(y\right)d^{2}+\left(1-y\right)\max\left(margin-d,0\right)^{2}$ where $d=0$	
$  a_n-b_n  _2$ . This can be used to train siamese networks caffe::ConvolutionLayer $<$ Dtype $>$	114
Convolves the input image with a bank of learned filters, and (optionally) adds biases	119
caffe::CPUTimer	124
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	126
caffe::db::Cursor	130
caffe::DataLayer< Dtype >	131
Applies common transformations to the input data, such as scaling, mirroring, substracting the	404
image mean	134
caffe::DB	138
Convolve the input with a bank of learned filters, and (optionally) add biases, treating filters and convolution parameters in the opposite sense as ConvolutionLayer	138
caffe::DropoutLayer< Dtype >	
During training only, sets a random portion of $x$ to 0, adjusting the rest of the vector magnitude accordingly	141
caffe::DummyDataLayer < Dtype > Provides data to the Net generated by a Filler	146
caffe::EltwiseLayer< Dtype >	
	151
Exponential Linear Unit non-linearity $y = \left\{ \begin{array}{ll} x & \text{if } x > 0 \\ \alpha(\exp(x) - 1) & \text{if } x \leq 0 \end{array} \right.$	155
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	159
caffe::EuclideanLossLayer< Dtype >	
Computes the Euclidean (L2) loss $E=rac{1}{2N}\sum\limits_{n=1}^{N}\left \left \hat{y}_{n}-y_{n}\right \right _{2}^{2}$ for real-valued regression tasks .	163
caffe::ExpLayer < Dtype > $\frac{\alpha x + \beta}{2}$ as a restited by the scale $\frac{\alpha x + \beta}{2}$ and here	400
Computes $y=\gamma^{\alpha x+\beta}$ , as specified by the scale $\alpha$ , shift $\beta$ , and base $\gamma$ caffe::Filler< Dtype >	169
Fills a Blob with constant or randomly-generated data	174
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)	175
caffe::FlattenLayer< Dtype > Reshapes the input Blob into flat vectors	181
caffe::GaussianFiller< Dtype >	100
Fills a Blob with Gaussian-distributed values $x=a$	186
caffe::HDF5DataLayer< Dtype >	188
Provides data to the Net from HDF5 files	188

3.1 Class List

caffe::HDF5OutputLayer< Dtype >	
Write blobs to disk as HDF5 files	194
caffe::HingeLossLayer< Dtype >	
Computes the hinge loss for a one-of-many classification task	199
caffe::Im2colLayer< Dtype >	
A helper for image operations that rearranges image regions into column vectors. Used by	
ConvolutionLayer to perform convolution by matrix multiplication	204
caffe::ImageDataLayer< Dtype >	
Provides data to the Net from image files	209
caffe::InfogainLossLayer< Dtype >	
A generalization of SoftmaxWithLossLayer that takes an "information gain" (infogain) matrix	
specifying the "value" of all label pairs	213
caffe::InnerProductLayer< Dtype >	
Also known as a "fully-connected" layer, computes an inner product with a set of learned weights,	
and (optionally) adds biases	221
caffe::InputLayer< Dtype >	
Provides data to the Net by assigning tops directly	226
caffe::InternalThread	
caffe::Layer< Dtype >	20
	233
An interface for the units of computation which can be composed into a Net	
caffe::LayerRegisterer< Dtype >	243
caffe::LayerRegistry < Dtype >	243
caffe::LogLayer< Dtype >	0.4
Computes $y = log_{\gamma}(\alpha x + \beta)$ , as specified by the scale $\alpha$ , shift $\beta$ , and base $\gamma$	244
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	249
caffe::LRNLayer< Dtype >	
Normalize the input in a local region across or within feature maps	253
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	258
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	262
caffe::MemoryDataLayer< Dtype >	
Provides data to the Net from memory	267
caffe::MSRAFiller< Dtype >	
Fills a Blob with values $x \sim N(0, \sigma^2)$ where $\sigma^2$ is set inversely proportional to number of	
incoming nodes, outgoing nodes, or their average	270
caffe::MultinomialLogisticLossLayer< Dtype >	
Computes the multinomial logistic loss for a one-of-many classification task, directly taking a	
predicted probability distribution as input	272
caffe::MVNLayer< Dtype >	
Normalizes the input to have 0-mean and/or unit (1) variance	277
caffe::NesterovSolver< Dtype >	
caffe::Net< Dtype >	
Connects Layers together into a directed acyclic graph (DAG) specified by a NetParameter	284
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	289
caffe::ParameterLayer< Dtype >	293
caffe::PoolingLayer < Dtype >	
Pools the input image by taking the max, average, etc. within regions	296
caffe::PositiveUnitballFiller< Dtype >	_00
Fills a Blob with values $x \in [0,1]$ such that $\forall i \sum_j x_{ij} = 1$	301
caffe::PowerLayer< Dtype >	50
Computes $y = (\alpha x + \beta)^{\gamma}$ , as specified by the scale $\alpha$ , shift $\beta$ , and power $\gamma$	303
$y = (\alpha x + \beta)$ , the specimen by the scale $\alpha$ , shift $\beta$ , and power $\gamma$	500

10 Class Index

caffe::PReLULayer< Dtype >	
Parameterized Rectified Linear Unit non-linearity $y_i = \max(0, x_i) + a_i \min(0, x_i)$ . The differ-	
ences 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	308
${\sf caffe::PythonLayer} < {\sf Dtype} >  \dots  \dots  \dots  \dots  \dots  \dots  \dots  \dots  \dots $	315
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	318
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	325
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	330
caffe::ReshapeLayer< Dtype >	335
caffe::RMSPropSolver< Dtype >	340
caffe::Caffe::RNG	342
	342
caffe::RNNLayer< Dtype >	
Processes time-varying inputs using a simple recurrent neural network (RNN). Implemented as a network unrolling the RNN computation in time	343
·	343
caffe::ScaleLayer< Dtype >	
Computes the elementwise product of two input Blobs, with the shape of the latter Blob "broad-	
cast" 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	0.45
"broadcast" sum too when bias_term: true is set	345
caffe::SGDSolver< Dtype >	054
Optimizes the parameters of a Net using stochastic gradient descent (SGD) with momentum .	351
caffe::SigmoidCrossEntropyLossLayer< Dtype >	
Computes the cross-entropy (logistic) loss $E=rac{-1}{n}\sum_{n=1}^{N}\left[p_{n}\log\hat{p}_{n}+(1-p_{n})\log(1-\hat{p}_{n})\right]$ , of-	
ten used for predicting targets interpreted as probabilities	353
caffe::SigmoidLayer< Dtype >	
Sigmoid function non-linearity $y=(1+\exp(-x))^{-1}$ , a classic choice in neural networks	359
caffe::SignalHandler	363
caffe::SilenceLayer< Dtype >	
Ignores bottom blobs while producing no top blobs. (This is useful to suppress outputs during	
testing.)	364
caffe::SliceLayer< Dtype >	
Takes a Blob and slices it along either the num or channel dimension, outputting multiple sliced	
Blob results	367
caffe::SoftmaxLayer< Dtype >	
Computes the softmax function	373
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	378
caffe::Solver< Dtype >	
An interface for classes that perform optimization on Nets	
${\sf caffe::} {\sf SolverRegisterer} {\sf < Dtype} {\sf > } \dots \dots$	
${\sf caffe::} {\sf SolverRegistry} {\sf < Dtype} {\sf > } \ldots \ldots$	388
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	389
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	393

3.1 Class List

caffe::SwishLayer< Dtype >	
Swish non-linearity $y=x\sigma(\beta x)$ . A novel activation function that tends to work better than ReLU	
[1]	398
$caffe:: Blocking Queue < T > :: sync \\  \   \dots \\ \  \   \dots$	404
caffe::SyncedMemory	
Manages memory allocation and synchronization between the host (CPU) and device (GPU) .	404
caffe::TanHLayer< Dtype >	
TanH hyperbolic tangent non-linearity $y=rac{\exp(2x)-1}{\exp(2x)+1}$ , popular in auto-encoders	405
caffe::ThresholdLayer< Dtype >	
Tests whether the input exceeds a threshold: outputs 1 for inputs above threshold; 0 otherwise	409
caffe::TileLayer< Dtype >	
Copy a Blob along specified dimensions	413
caffe::Timer	418
caffe::db::Transaction	420
caffe::UniformFiller< Dtype >	
Fills a Blob with uniformly distributed values $x \sim U(a,b) - \ldots - \ldots - \ldots$	421
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	422
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	426

12 Class Index

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

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  $E = \frac{1}{2N} \sum_{n=1}^{N} (y) d^2 + (1-y) \max (margin - d, 0)^2$  where  $d = ||a_n - b_n||_2$ . 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 
$$y=\left\{ egin{array}{ll} x & \mbox{if } x>0 \\ \alpha(\exp(x)-1) & \mbox{if } x\leq0 \end{array} \right.$$

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  $E = \frac{1}{2N} \sum_{n=1}^{N} ||\hat{y}_n - y_n||_2^2$  for real-valued regression tasks.

class ExpLayer

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

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 LaverRegisterer
- class LayerRegistry
- class LogLayer

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

· 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  $\sigma^2$  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  $\forall i \sum_{i} x_{ij} = 1$ .

class PowerLayer

Computes  $y = (\alpha x + \beta)^{\gamma}$ , as specified by the scale  $\alpha$ , shift  $\beta$ , and power  $\gamma$ .

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  $E = \frac{-1}{n} \sum_{n=1}^{N} [p_n \log \hat{p}_n + (1-p_n) \log (1-\hat{p}_n)]$ , 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  $y=rac{\exp(2x)-1}{\exp(2x)+1}$ , 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 col\_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 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)

 $\bullet \quad {\sf template}{<} {\sf typename \ Dtype} >$ 

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)

 $\bullet \quad {\sf template}{<} {\sf typename} \ {\sf Dtype}>$ 

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)

 $\bullet \quad {\sf template}{<} {\sf typename\ Dtype} >$ 

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\_gpu\_exp (const int n, const Dtype \*a, Dtype \*y)

 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<tvpename Dtvpe > 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\_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)

 $\bullet \quad {\sf template}{<} {\sf typename\ Dtype} >$ 

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)

 $\bullet \quad \mathsf{template}{<}\mathsf{class}\;\mathsf{RandomAccessIterator}>$ 

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)
- ullet 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)
   PEOLOTER LAYER OF ACC (Flatter)
- 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 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 \*q, 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\_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)
  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 btype \* v
- 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()

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).

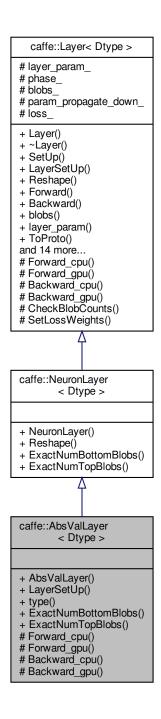
# **Chapter 5**

# **Class Documentation**

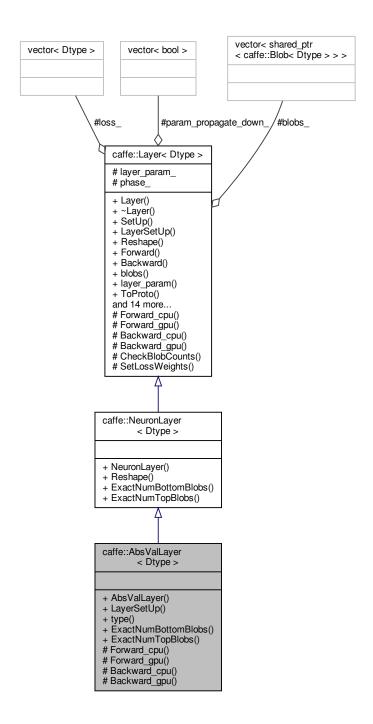
 ${\bf 5.1} \quad {\bf caffe::AbsValLayer} < {\bf Dtype} > {\bf Class\ Template\ Reference}$ 

Computes y = |x|.

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()

#### **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()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, 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

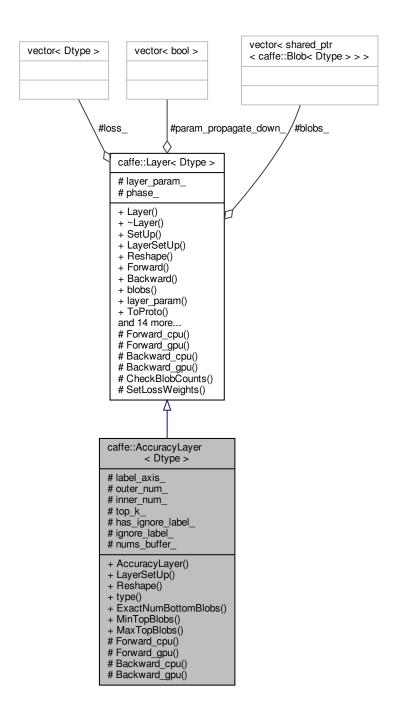
Computes the classification accuracy for a one-of-many classification task.

```
#include <accuracy_layer.hpp>
```

Inheritance diagram for caffe::AccuracyLayer< Dtype >:

```
caffe::Layer< Dtype >
# layer_param_
# phase
# blobs_
# param_propagate_down_
# İoss
+ Layer()
+ ~Layer()
+ SetÚp()
+ LayerSetUp()
+ Reshape()
+ Forward()
+ Backward()
+ blobs()
+ layer_param()
+ ToProto()
and 14 more...
# Forward_cpu()
# Forward_gpu()
# Backward_cpu()
# Backward_gpu()
# CheckBlobCounts()
# SetLossWeights()
caffe::AccuracyLayer
         < Dtype >
# label_axis_
# outer_num_
# inner_num_
# top_k_
# has_ignore_label_
# ignore label
# nums_buffer_
+ AccuracyLayer()
+ LayerSetUp()
+ Reshape()
+ type()
+ ExactNumBottomBlobs()
+ MinTopBlobs()
+ MaxTopBlobs()
# Forward_cpu()
# Forward_gpu()
# Backward_cpu()
# Backward_gpu()
```

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()

#### **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()

#### **Parameters**

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: $\hat{l}_n = \arg\max_k x_{nk}$		
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		
output Blob vector (length 1)		
1. $(1 \times 1 \times 1 \times 1)$ the computed accuracy: $\frac{1}{N} \sum_{n=1}^{N} \delta\{\hat{l}_n = l_n\}$ , where $\delta\{\text{condition}\} = \begin{cases} 1 & \text{if condition} \\ 0 & \text{otherwise} \end{cases}$		

Implements caffe::Layer< Dtype >.

#### 5.2.3.3 LayerSetUp()

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

#### **Parameters**

bottom	<ul><li>the preshaped input blobs, whose data fields store the input data for this lay</li><li>the allocated but unshaped output blobs</li></ul>	
top		

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, 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()

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.

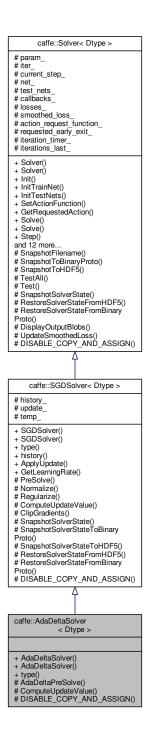
Implements caffe::Layer< Dtype >.

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

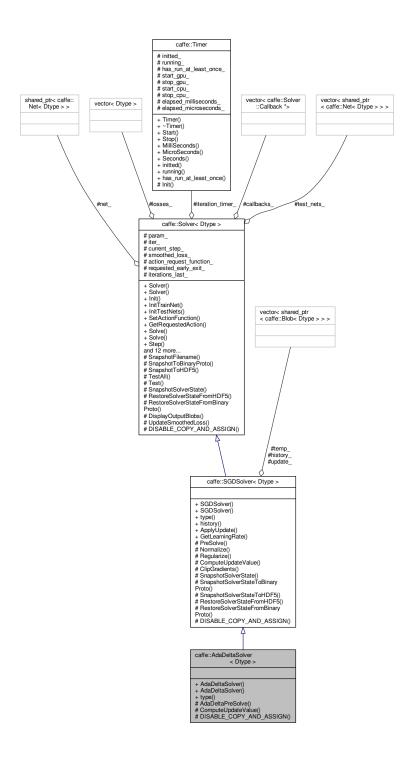
- include/caffe/layers/accuracy\_layer.hpp
- src/caffe/layers/accuracy\_layer.cpp

# 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)

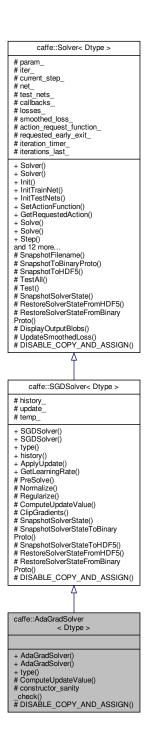
# **Additional Inherited Members**

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

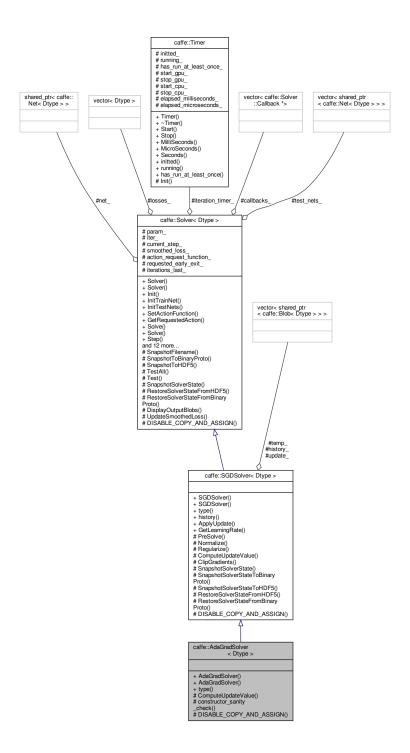
- include/caffe/sgd\_solvers.hpp
- src/caffe/solvers/adadelta\_solver.cpp

# 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**

- void constructor\_sanity\_check ()
- DISABLE\_COPY\_AND\_ASSIGN (AdaGradSolver)

# **Additional Inherited Members**

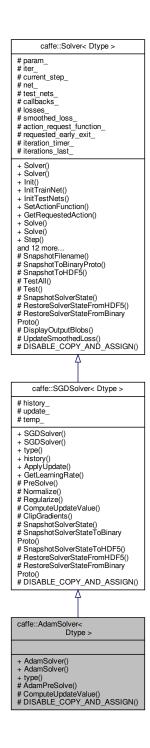
The documentation for this class was generated from the following files:

- include/caffe/sgd\_solvers.hpp
- src/caffe/solvers/adagrad\_solver.cpp

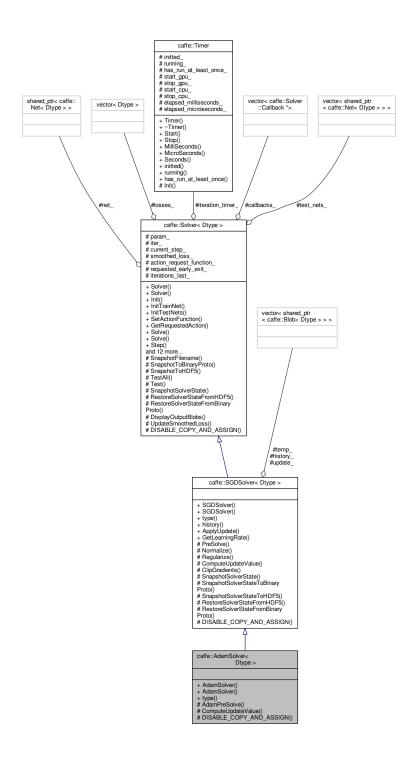
# 5.5 caffe::AdamSolver < Dtype > Class Template Reference

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

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

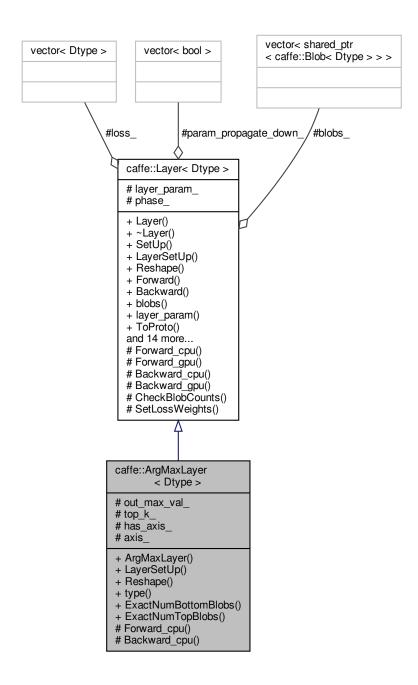
Compute the index of the K max values for each datum across all dimensions  $(C \times H \times W)$ .

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

Inheritance diagram for caffe::ArgMaxLayer< Dtype >:

```
caffe::Layer< Dtype >
# layer_param_
# phase
# blobs_
# param_propagate_down_
# loss_
+ Layer()
+ ~Layer()
+ SetUp()
+ LayerSetUp()
+ Reshape()
+ Forward()
+ Backward()
+ blobs()
+ layer_param()
+ ToProto()
and 14 more...
# Forward_cpu()
# Forward_gpu()
# Backward_cpu()
# Backward_gpu()
# CheckBlobCounts()
# SetLossWeights()
caffe::ArgMaxLayer
        < Dtype >
# out_max_val_
# top_k_
# has_axis_
# axis
+ ArgMaxLayer()
+ LayerSetUp()
+ Reshape()
+ type() + ExactNumBottomBlobs()
+ ExactNumTopBlobs()
# Forward_cpu()
# Backward_cpu()
```

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()

### **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()

### **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. $(N \times K \times K)$ 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()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

# 5.6.3.5 Reshape()

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

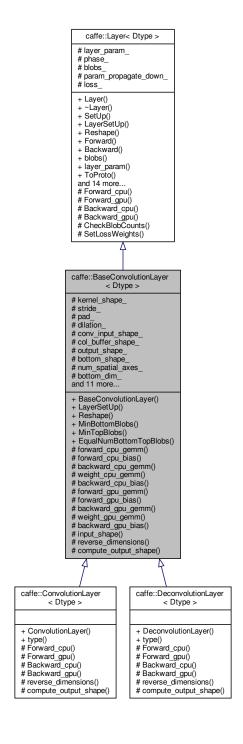
## **Parameters**

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

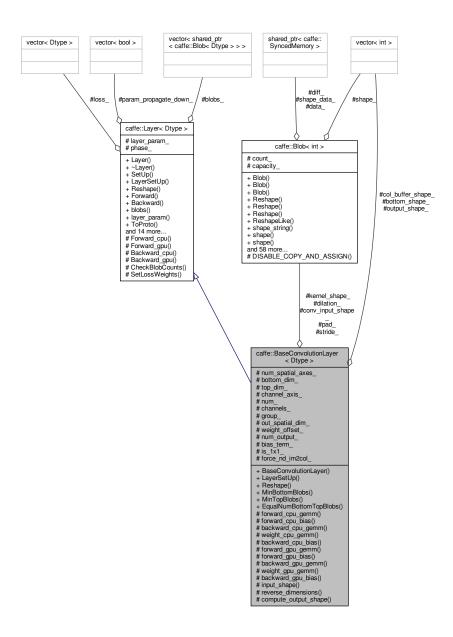
This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.
Implements caffe::Layer < Dtype >.
The documentation for this class was generated from the following files:
<ul> <li>include/caffe/layers/argmax_layer.hpp</li> <li>src/caffe/layers/argmax_layer.cpp</li> </ul>
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

```
\label{template} \mbox{template}{<} \mbox{typename Dtype}{>} \\ \mbox{class caffe::BaseConvolutionLayer}{<} \mbox{ 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()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, 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()

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:

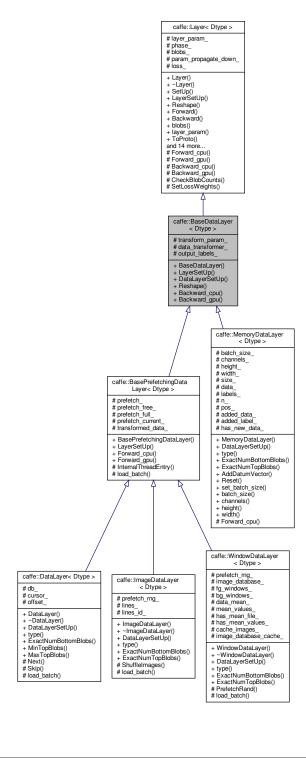
- include/caffe/layers/base\_conv\_layer.hpp
- src/caffe/layers/base\_conv\_layer.cpp

# 5.8 caffe::BaseDataLayer < Dtype > Class Template Reference

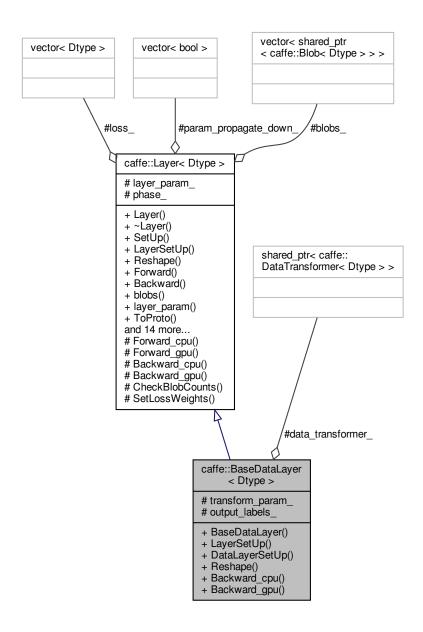
Provides base for data layers that feed blobs to the Net.

```
#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

```
\label{template} \mbox{typename Dtype} > \\ \mbox{class caffe::BaseDataLayer} < \mbox{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()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

 $\label{lem:basePrefetchingDataLayer} Reimplemented \ in \ caffe:: BasePrefetchingDataLayer < \ D type >.$ 

# 5.8.2.2 Reshape()

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.

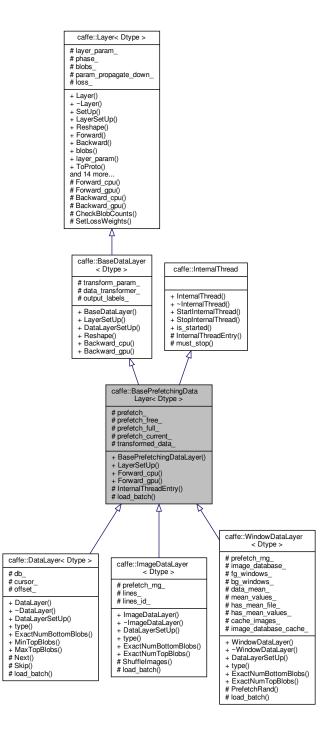
Implements caffe::Layer< Dtype >.

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

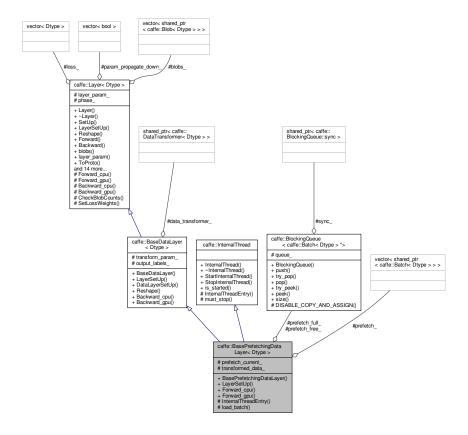
- include/caffe/layers/base\_data\_layer.hpp
- src/caffe/layers/base\_data\_layer.cpp

# 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()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, which will be called before the forward pass to adjust the top blob sizes.

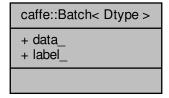
Reimplemented from caffe::BaseDataLayer< Dtype >.

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

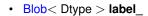
# 5.10 caffe::Batch < Dtype > Class Template Reference

Collaboration diagram for caffe::Batch < Dtype >:



<b>Public</b>	<b>Attributes</b>
---------------	-------------------





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

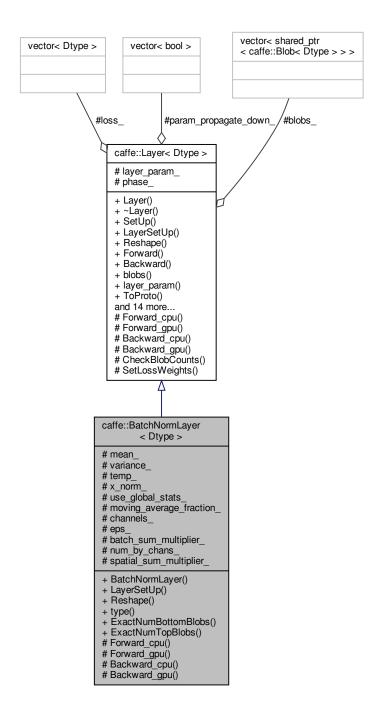
Normalizes the input to have 0-mean and/or unit (1) variance across the batch.

#include <batch\_norm\_layer.hpp>

Inheritance diagram for caffe::BatchNormLayer< Dtype >:

```
caffe::Layer< Dtype >
 # layer_param_
 # phase_
 # blobs
 # param_propagate_down_
# loss_
 + Layer()
+ ~Layer()
+ SetUp()
+ LayerSetUp()
+ Reshape()
 + Forward()
 + Backward()
 + blobs()
 + layer_param()
+ ToProto()
 and 14 more...
 # Forward_cpu()
# Forward_gpu()
# Backward_cpu()
 # Backward_gpu()
 # CheckBlobCounts()
 # SetLossWeights()
caffe::BatchNormLayer
            < Dtype >
# mean
# variance_
# temp_
# x_norm
# use_global_stats_
# moving_average_fraction_
# channels_
# eps_
# batch_sum_multiplier_
# num_by_chans_
# spatial_sum_multiplier_
+ BatchNormLayer()
+ LayerSetUp()
+ Reshape()
+ hestiape()
+ type()
+ ExactNumBottomBlobs()
+ ExactNumTopBlobs()
# Forward_cpu()
# Forward_gpu()
# Backward_cpu()
# Backward_gpu()
```

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 true after each BatchNormLayer to handle both the bias and scaling factor.

[1] S. loffe 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()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

### 5.11.2.4 Reshape()

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

### **Parameters**

botte	om	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

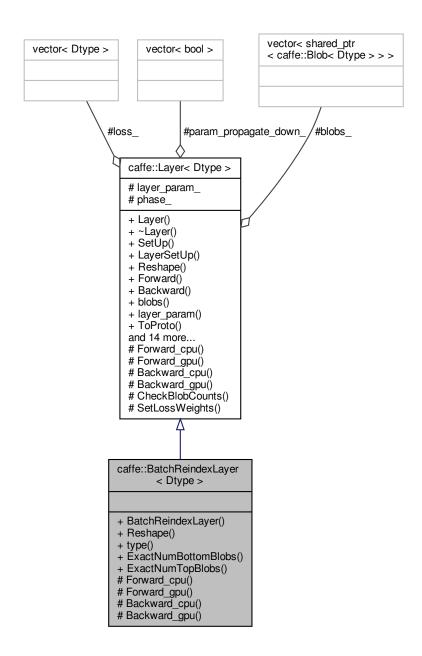
Index into the input blob along its first axis.

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

Inheritance diagram for caffe::BatchReindexLayer< Dtype >:

```
caffe::Layer< Dtype >
# layer_param_
# phase
# blobs_
# param_propagate_down_
# loss
+ Layer()
+ ~Layer()
+ SetUp()
+ LayerSetUp()
+ Reshape()
+ Forward()
+ Backward()
+ blobs()
+ layer_param()
+ ToProto()
and 14 more...
# Forward_cpu()
# Forward_gpu()
# Backward_cpu()
# Backward_gpu()
# CheckBlobCounts()
# SetLossWeights()
caffe::BatchReindexLayer
        < Dtype >
+ BatchReindexLayer()
+ Reshape()
+ type()
+ ExactNumBottomBlobs()
+ ExactNumTopBlobs()
# Forward_cpu()
# Forward_gpu()
# Backward_cpu()
# Backward_gpu()
```

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()

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.  Generated by Doxyg  Oxyge  O

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()

### **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()

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

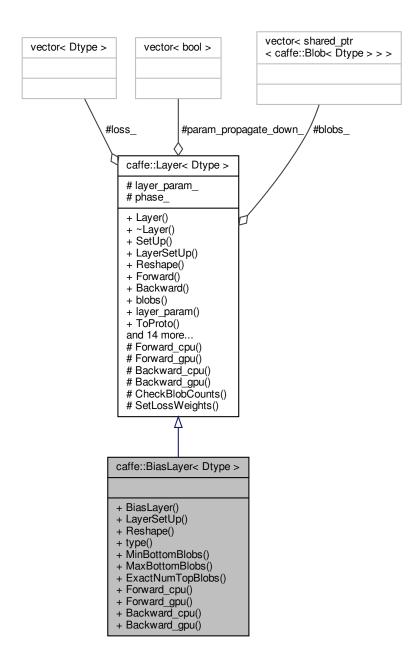
Computes a sum of two input Blobs, with the shape of the latter Blob "broadcast" to match the shape of the former. Equivalent to tiling the latter Blob, then computing the elementwise sum.

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

Inheritance diagram for caffe::BiasLayer< Dtype >:

# caffe::Layer< Dtype > # layer\_param\_ # phase # blobs\_ # param\_propagate\_down\_ # İoss + Layer() + ~Layer() + SetÚp() + LayerSetUp() + Reshape() + Forward() + Backward() + blobs() + layer\_param() + ToProto() and 14 more... # Forward\_cpu() # Forward\_gpu() # Backward\_cpu() # Backward\_gpu() # CheckBlobCounts() # SetLossWeights() caffe::BiasLayer< Dtype > + BiasLayer() + LayerSetUp() + Reshape() + type() + MinBottomBlobs() + MaxBottomBlobs() + ExactNumTopBlobs() + Forward\_cpu() + Forward\_gpu() + Backward\_cpu() + Backward\_gpu()

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

```
\label{lem:continuous} \mbox{template}{<} \mbox{typename Dtype}{>} \\ \mbox{class caffe::BiasLayer}{<} \mbox{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()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, 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()

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:

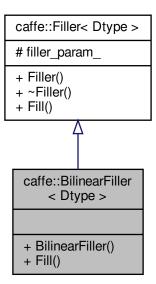
- include/caffe/layers/bias\_layer.hpp
- src/caffe/layers/bias\_layer.cpp

# 5.14 caffe::BilinearFiller < Dtype > Class Template Reference

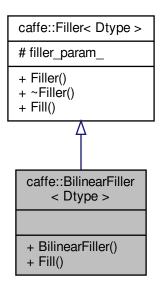
Fills a Blob with coefficients for bilinear interpolation.

#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:

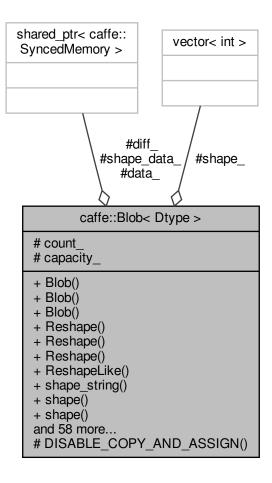
• include/caffe/filler.hpp

# 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()

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]

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]

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()

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()

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()

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.

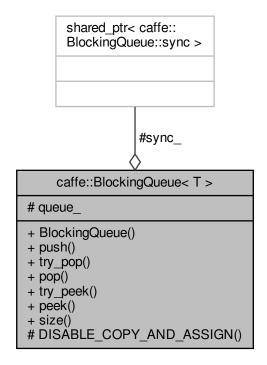
This deallocates the SyncedMemory holding this Blob's diff\_, as shared\_ptr calls its destructor when reset with the "=" operator.

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:

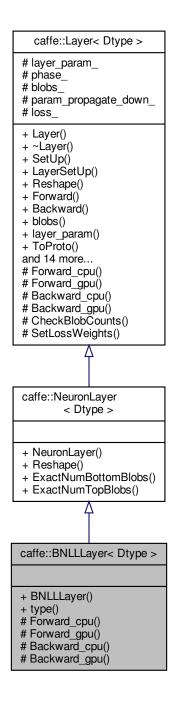
- include/caffe/util/blocking\_queue.hpp
- src/caffe/util/blocking\_queue.cpp

# 5.17 caffe::BNLLLayer < Dtype > Class Template Reference

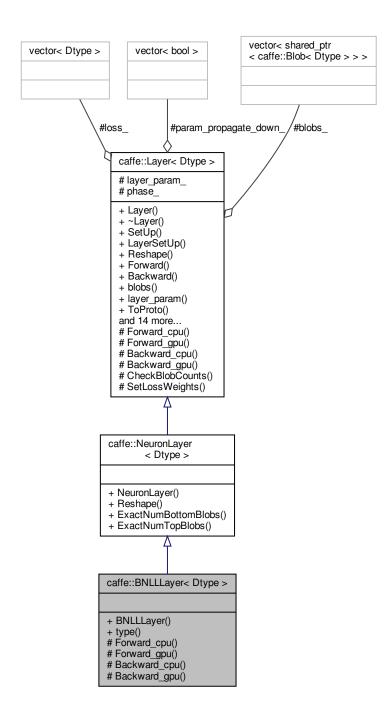
Computes  $y = x + \log(1 + \exp(-x))$  if x > 0;  $y = \log(1 + \exp(x))$  otherwise.

#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

```
\label{template} \mbox{template}{<} \mbox{typename Dtype}{>} \\ \mbox{class caffe::BNLLLayer}{<} \mbox{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 $y = \begin{cases} x + \log(1 + \exp(-x)) \\ \log(1 + \exp(x)) \end{cases}$	$\begin{array}{l} \text{if } x>0 \\ \text{otherwise} \end{array}$

## 5.17.2 Member Function Documentation

#### 5.17.2.1 Backward\_cpu()

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()

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 $y = \left\{ \begin{array}{ll} x + \log(1 + \exp(-x)) & \text{if } x > 0 \\ \log(1 + \exp(x)) & \text{otherwise} \end{array} \right.$	

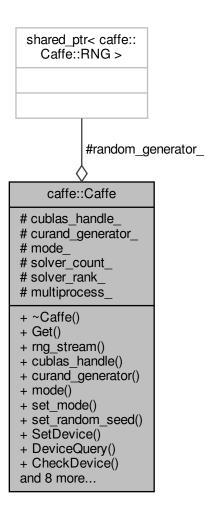
Implements caffe::Layer< Dtype >.

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**

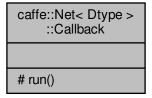
- cublasHandle\_t cublas\_handle\_
- curandGenerator\_t curand\_generator\_
- shared\_ptr< RNG > random\_generator\_
- Brew mode
- int solver\_count\_
- int solver\_rank\_
- · bool multiprocess\_

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

 $\label{lem:collaboration} \mbox{Collaboration diagram for caffe::Solver} < \mbox{Dtype} > :: \mbox{Callback:}$ 

caffe::Solver< Dtype
>::Callback

# on\_start()
# on\_gradients\_ready()

## **Protected Member Functions**

- virtual void on\_start ()=0
- virtual void on\_gradients\_ready ()=0

## **Friends**

template<typename T > class Solver

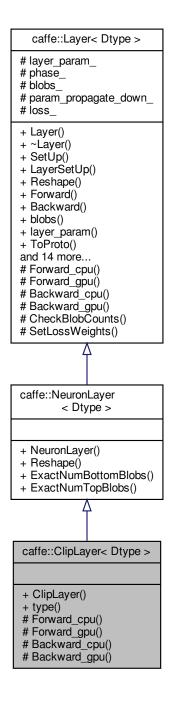
The documentation for this class was generated from the following file:

include/caffe/solver.hpp

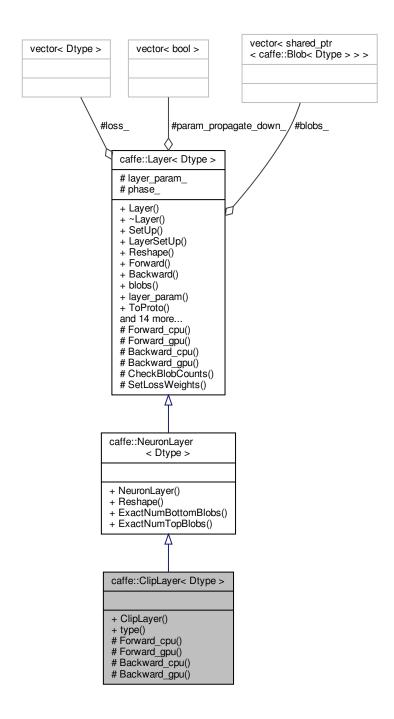
# 5.21 caffe::ClipLayer < Dtype > Class Template Reference

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()

## Parameters

```
param provides ClipParameter clip_param, with ClipLayer options:

• min

• max
```

#### 5.21.3 Member Function Documentation

#### 5.21.3.1 Backward\_cpu()

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 $\frac{\partial E}{\partial x} = \left\{ \begin{array}{ll} 0 & \text{if } x < min \lor x > max \\ \frac{\partial E}{\partial y} & \text{if } x \geq min \land x \leq max \end{array} \right.$

Implements caffe::Layer< Dtype >.

#### 5.21.3.2 Forward\_cpu()

#### **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:

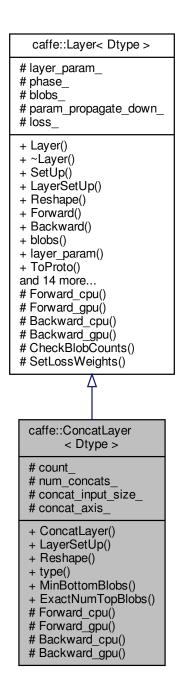
- include/caffe/layers/clip\_layer.hpp
- src/caffe/layers/clip\_layer.cpp

# 5.22 caffe::ConcatLayer < Dtype > Class Template Reference

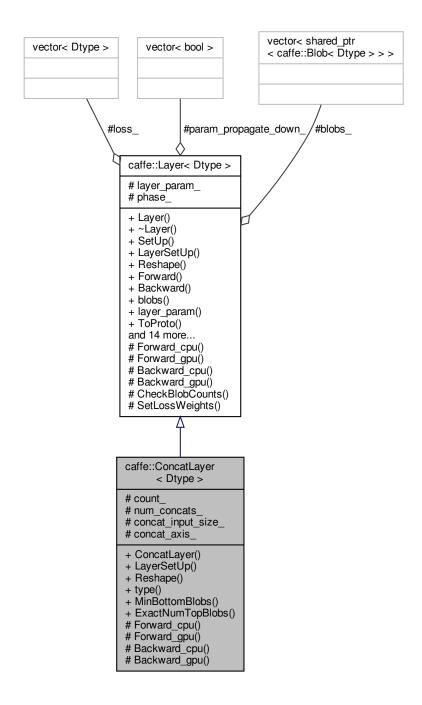
Takes at least two Blobs and concatenates them along either the num or channel dimension, outputting the result.

```
#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()

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 $\begin{bmatrix} \frac{\partial E}{\partial x_1} & \frac{\partial E}{\partial x_2} & \dots & \frac{\partial E}{\partial x_K} \end{bmatrix} = \frac{\partial E}{\partial y}$

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()

#### **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 $y = [\begin{array}{ccc} x_1 & x_2 & \dots & x_K \end{array}]$

Implements caffe::Layer< Dtype >.

#### 5.22.2.4 LayerSetUp()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, 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()

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:

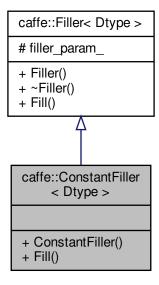
- include/caffe/layers/concat\_layer.hpp
- src/caffe/layers/concat\_layer.cpp

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

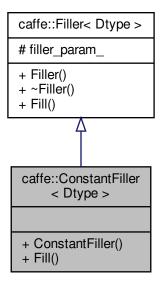
Fills a Blob with constant values x = 0.

#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:

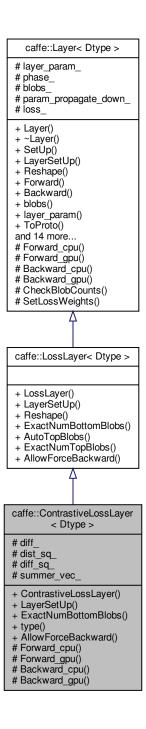
include/caffe/filler.hpp

# 5.24 caffe::ContrastiveLossLayer < Dtype > Class Template Reference

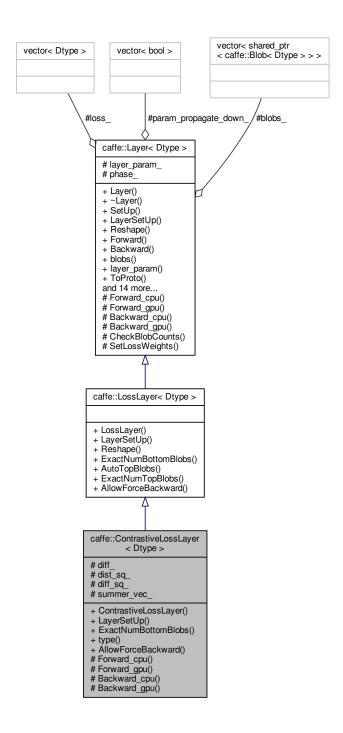
Computes the contrastive loss  $E = \frac{1}{2N} \sum_{n=1}^{N} (y) d^2 + (1-y) \max (margin - d, 0)^2$  where  $d = ||a_n - b_n||_2$ . This can be used to train siamese networks.

#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  $E = \frac{1}{2N} \sum_{n=1}^{N} (y) d^2 + (1-y) \max (margin - d, 0)^2$  where  $d = ||a_n - b_n||_2$ . 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  $E = \frac{1}{2N} \sum_{n=1}^{N} (y) d^2 + (1-y) \max (margin - d, 0)^2$  where  $d = ||a_n - b_n||_2$ . 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. $(N  imes C  imes 1  imes 1)$ 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:
	$E = \frac{1}{2N} \sum_{n=1}^{N} (y) d^2 + (1-y) \max (margin - d, 0)^2$ where $d =   a_n - b_n  _2$ . This can be
	used to train siamese networks.

## 5.24.2 Member Function Documentation

#### 5.24.2.1 AllowForceBackward()

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()

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 $ \begin{tabular}{ll} 1. & (1\times 1\times 1\times 1) \begin{tabular}{ll} This Blob's diff will simply contain the loss_weight* $\lambda$, as $\lambda$ is the coefficient of this layer's output $\ell_i$ 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.)   \end{tabular} $
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()

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

Implements caffe::Layer< Dtype >.

## 5.24.2.5 LayerSetUp()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, 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:

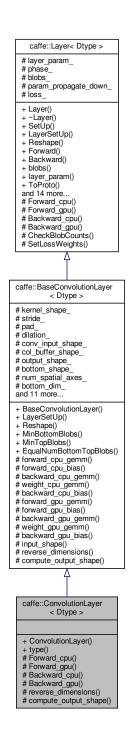
- include/caffe/layers/contrastive\_loss\_layer.hpp
- src/caffe/layers/contrastive\_loss\_layer.cpp

# 5.25 caffe::ConvolutionLayer < Dtype > Class Template Reference

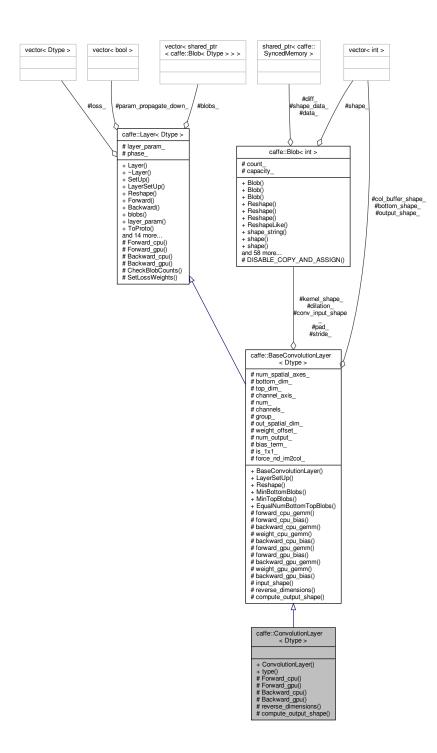
Convolves the input image with a bank of learned filters, and (optionally) adds biases.

#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()

#### **Parameters**

param

provides ConvolutionParameter convolution param, with ConvolutionLayer options:

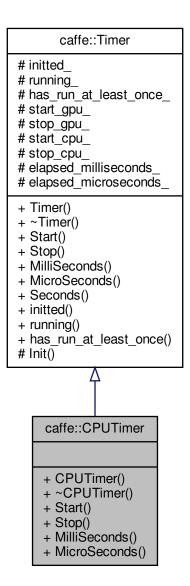
- · num\_output. The number of filters.
- kernel\_size / kernel\_h / kernel\_w. The filter dimensions, given by kernel\_size for square filters
  or kernel\_h and kernel\_w for rectangular filters.
- stride / stride\_h / stride\_w (**optional**, default 1). The filter stride, given by stride\_size for 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:

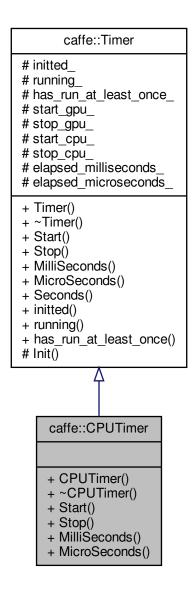
- include/caffe/layers/conv\_layer.hpp
- src/caffe/layers/conv\_layer.cpp

## 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:

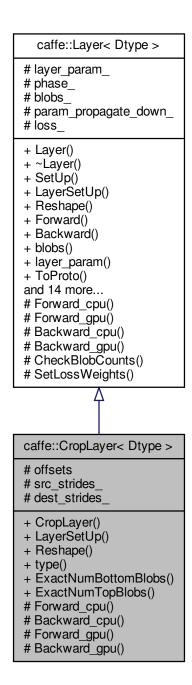
- include/caffe/util/benchmark.hpp
- src/caffe/util/benchmark.cpp

## 5.27 caffe::CropLayer < Dtype > Class Template Reference

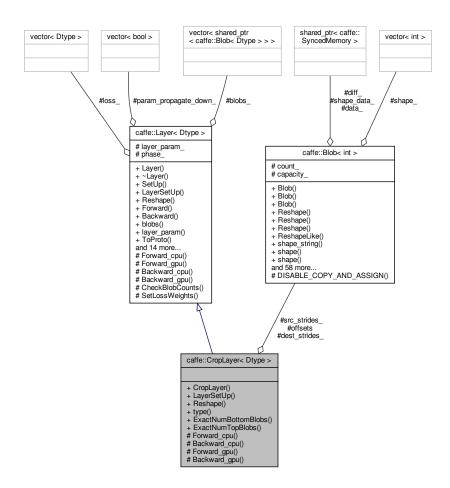
Takes a Blob and crop it, to the shape specified by the second input Blob, across all dimensions after the specified axis.

```
#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()

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

#### **Parameters**

botto	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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

## 5.27.2.4 Reshape()

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:

caffe::db::Cursor

+ Cursor()
+ ~Cursor()
+ SeekToFirst()
+ Next()
+ key()
+ value()
+ valid()
+ DISABLE\_COPY\_AND\_ASSIGN()

## **Public Member Functions**

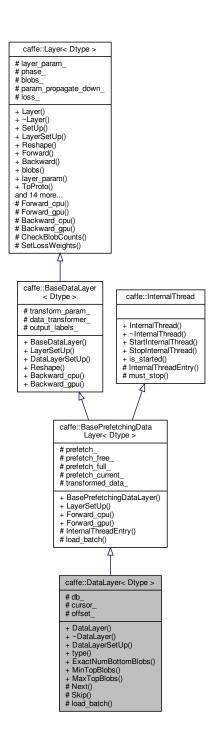
- virtual void SeekToFirst ()=0
- virtual void Next ()=0
- virtual string **key** ()=0
- virtual string value ()=0
- virtual bool valid ()=0
- · DISABLE COPY AND ASSIGN (Cursor)

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

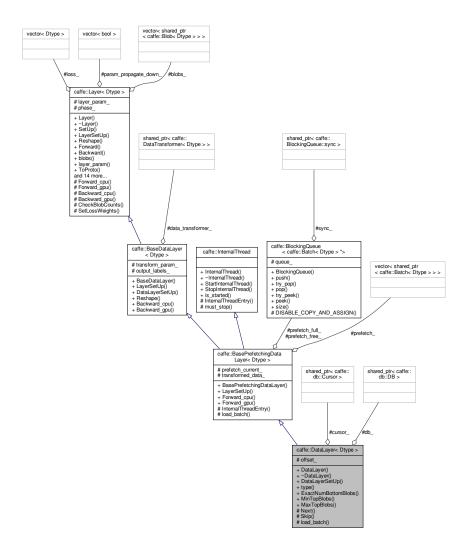
include/caffe/util/db.hpp

## 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:

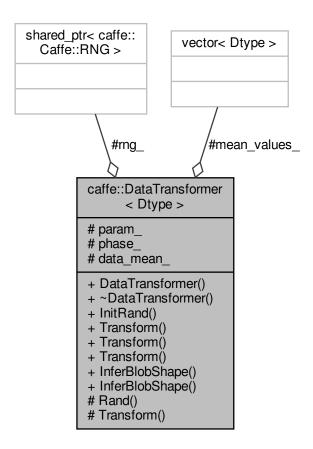
- include/caffe/layers/data\_layer.hpp
- src/caffe/layers/data\_layer.cpp

## 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]
```

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]

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**

dati	um_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]

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:

caffe::db::DB

+ DB()
+ ~DB()
+ Open()
+ Close()
+ NewCursor()
+ NewTransaction()
+ DISABLE\_COPY\_AND\_ASSIGN()

## **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:

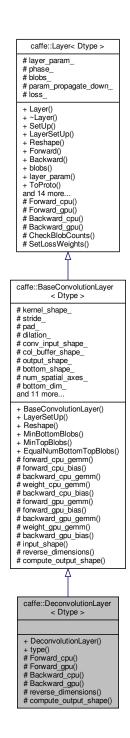
• include/caffe/util/db.hpp

## 5.32 caffe::DeconvolutionLayer < Dtype > Class Template Reference

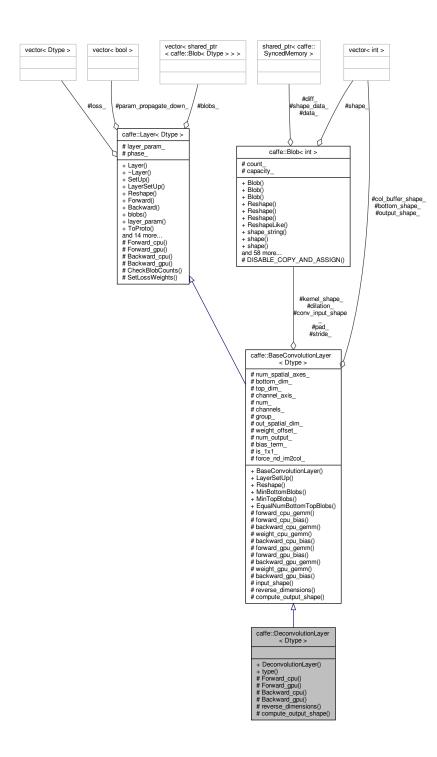
Convolve the input with a bank of learned filters, and (optionally) add biases, treating filters and convolution parameters in the opposite sense as ConvolutionLayer.

#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:

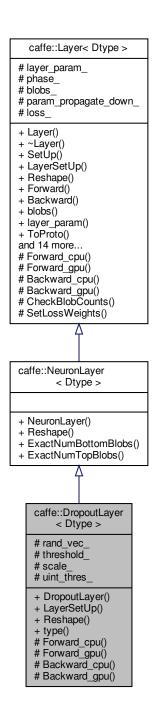
- include/caffe/layers/deconv layer.hpp
- src/caffe/layers/deconv\_layer.cpp

## 5.33 caffe::DropoutLayer < Dtype > Class Template Reference

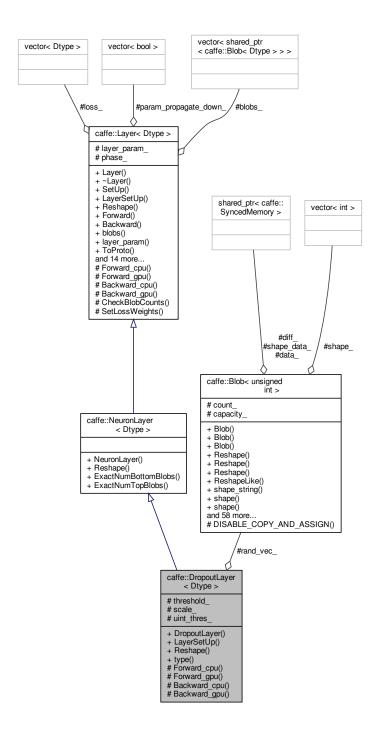
During training only, sets a random portion of x to 0, adjusting the rest of the vector magnitude accordingly.

```
#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()

#### **Parameters**

param	provides DropoutParameter dropout_param, with DropoutLayer options:
	- dropout_ratio ( <b>optional</b> , 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()

## **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 $y_{train} = \left\{ egin{array}{l} \frac{x}{1-p} & \text{if } u > p \\ 0 & \text{otherwise} \end{array} \right.$ , where $u \sim U(0,1)$ is generated independently for each input at each iteration. At test time, we simply have $y_{test} = \mathbb{E}[y_{train}] = x$ .

Implements caffe::Layer< Dtype >.

## 5.33.3.2 LayerSetUp()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

#### 5.33.3.3 Reshape()

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

## **Parameters**

botte	om	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.

 $\label{lem:new_problem} \mbox{Reimplemented from caffe::} \mbox{NeuronLayer} < \mbox{Dtype} >.$ 

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

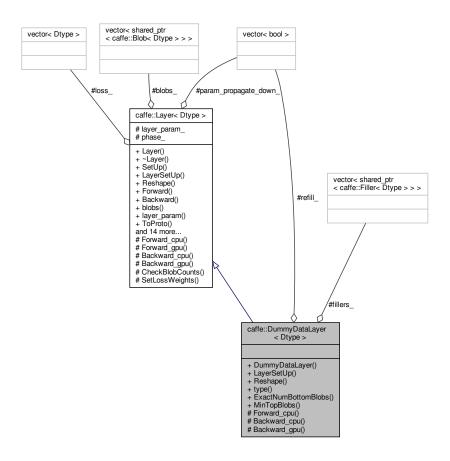
Provides data to the Net generated by a Filler.

```
#include <dummy_data_layer.hpp>
```

Inheritance diagram for caffe::DummyDataLayer< Dtype >:

## caffe::Layer< Dtype > # layer\_param\_ # phase # blobs\_ # param\_propagate\_down\_ # İoss + Layer() + ~Layer() + SetÚp() + LayerSetUp() + Reshape() + Forward() + Backward() + blobs() + layer\_param() + ToProto() and 14 more... # Forward\_cpu() # Forward\_gpu() # Backward\_cpu() # Backward\_gpu() # CheckBlobCounts() # SetLossWeights() caffe::DummyDataLayer < Dtype > # fillers\_ # refill\_ + DummyDataLayer() + LayerSetUp() + Reshape() + type() + ExactNumBottomBlobs() + MinTopBlobs() # Forward\_cpu() # Backward\_cpu() # Backward\_gpu()

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()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, 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()

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:

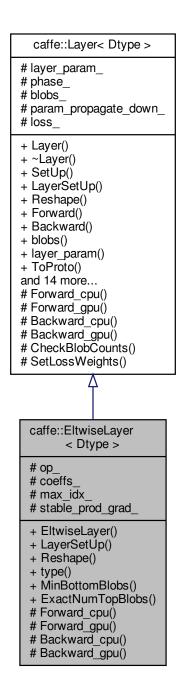
- include/caffe/layers/dummy\_data\_layer.hpp
- src/caffe/layers/dummy\_data\_layer.cpp

## 5.35 caffe::EltwiseLayer < Dtype > Class Template Reference

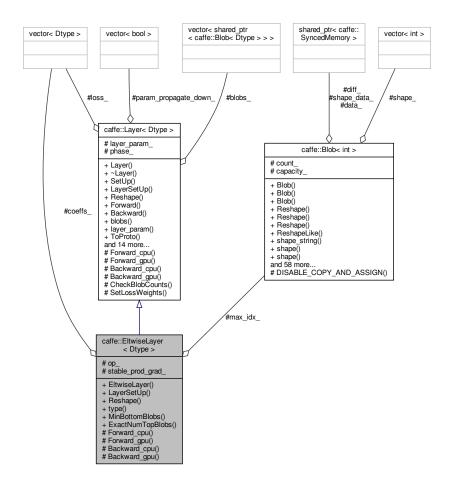
Compute elementwise operations, such as product and sum, along multiple input Blobs.

#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()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, 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()

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:

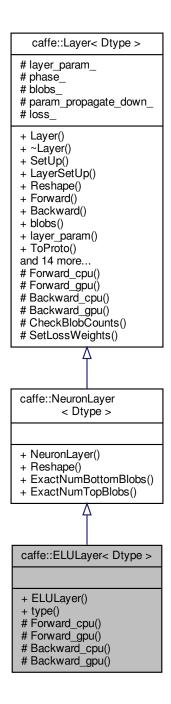
- include/caffe/layers/eltwise\_layer.hpp
- src/caffe/layers/eltwise\_layer.cpp

## 5.36 caffe::ELULayer < Dtype > Class Template Reference

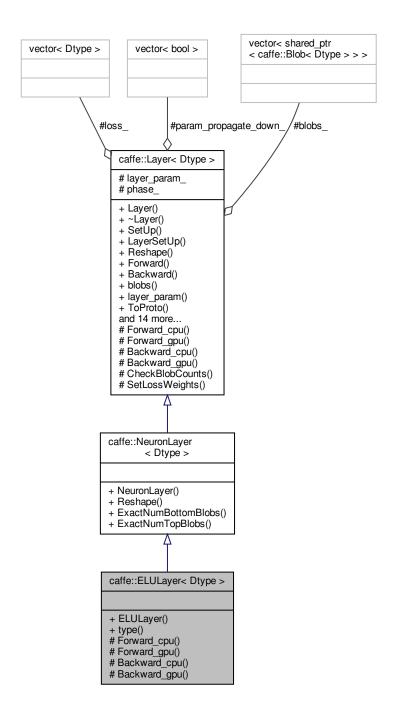
Exponential Linear Unit non-linearity  $y=\left\{ egin{array}{ll} x & \mbox{if } x>0 \\ \alpha(\exp(x)-1) & \mbox{if } x\leq 0 \end{array} 
ight. .$ 

#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 
$$y=\left\{ \begin{array}{ll} x & \text{if } x>0 \\ \alpha(\exp(x)-1) & \text{if } x\leq 0 \end{array} \right.$$

## 5.36.2 Constructor & Destructor Documentation

## 5.36.2.1 ELULayer()

## Parameters

param | provides ELUParameter elu\_param, with ELULayer options:

• alpha (**optional**, default 1). the value  $\alpha$  by which controls saturation for negative inputs.

## 5.36.3 Member Function Documentation

## 5.36.3.1 Backward\_cpu()

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 $\frac{\partial E}{\partial x} = \left\{ \begin{array}{ll} 1 & \text{if } x > 0 \\ y + \alpha & \text{if } x \leq 0 \end{array} \right.$ if propagate_down[0].

Implements caffe::Layer< Dtype >.

## 5.36.3.2 Forward\_cpu()

## **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 $y = \left\{ \begin{array}{ll} x & \text{if } x > 0 \\ \alpha(\exp(x) - 1) & \text{if } x \leq 0 \end{array} \right.$

Implements caffe::Layer< Dtype >.

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

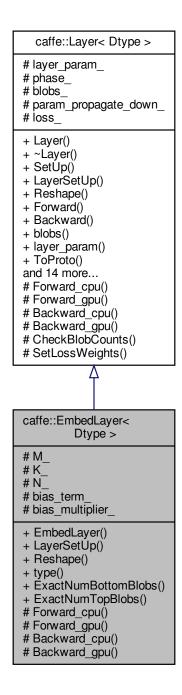
- include/caffe/layers/elu\_layer.hpp
- src/caffe/layers/elu\_layer.cpp

## 5.37 caffe::EmbedLayer < Dtype > Class Template Reference

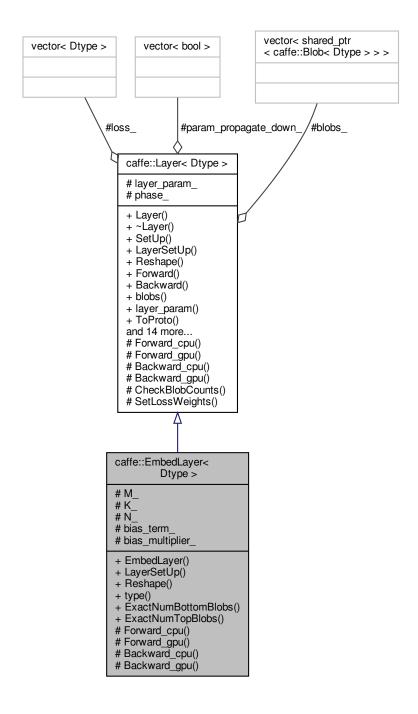
A layer for learning "embeddings" of one-hot vector input. Equivalent to an InnerProductLayer with one-hot vectors as input, but for efficiency the input is the "hot" index of each column itself.

```
#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()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

## 5.37.2.4 Reshape()

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:

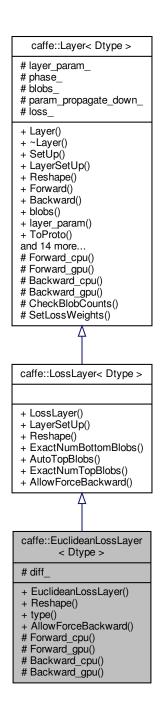
- include/caffe/layers/embed\_layer.hpp
- src/caffe/layers/embed\_layer.cpp

## 5.38 caffe::EuclideanLossLayer < Dtype > Class Template Reference

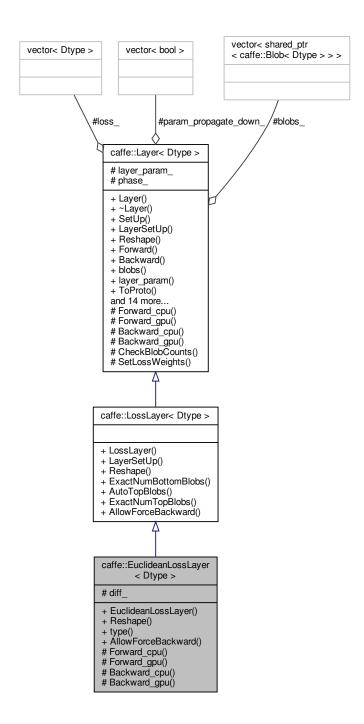
Computes the Euclidean (L2) loss  $E=\frac{1}{2N}\sum\limits_{n=1}^{N}||\hat{y}_n-y_n||_2^2$  for real-valued regression tasks.

#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  $E=\frac{1}{2N}\sum_{n=1}^{N}||\hat{y}_n-y_n||_2^2$  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  $E=\frac{1}{2N}\sum_{n=1}^{N}||\hat{y}_n-y_n||_2^2$  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: $E = \frac{1}{2n} \sum_{n=1}^{N}   \hat{y}_n - y_n  _2^2$

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()

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()

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* $\lambda$ , as $\lambda$ is the coefficient of this layer's output $\ell_i$ 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
	$rac{\partial E}{\partial \hat{y}} = rac{1}{n} \sum_{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
	$rac{\partial E}{\partial y} = rac{1}{n} \sum_{n=1}^{N} (y_n - \hat{y}_n)$ if propagate_down[1]

Implements caffe::Layer< Dtype >.

#### 5.38.2.3 Forward cpu()

Computes the Euclidean (L2) loss  $E=\frac{1}{2N}\sum\limits_{n=1}^{N}||\hat{y}_n-y_n||_2^2$  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: $E = \frac{1}{2n} \sum_{n=1}^{N}   \hat{y}_n - y_n  _2^2$

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()

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 ne	cessary 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:

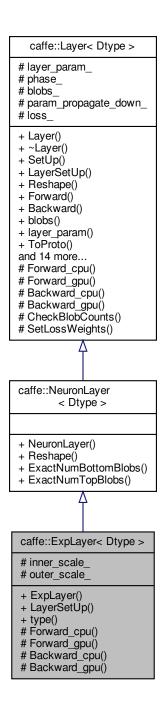
- include/caffe/layers/euclidean\_loss\_layer.hpp
- src/caffe/layers/euclidean\_loss\_layer.cpp

# 5.39 caffe::ExpLayer< Dtype > Class Template Reference

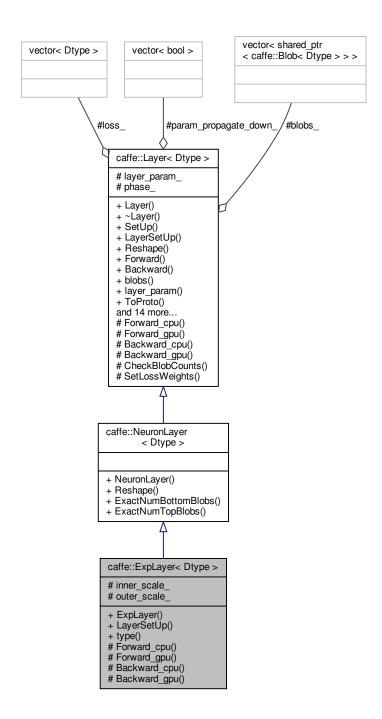
Computes  $y=\gamma^{\alpha x+\beta}$ , as specified by the scale  $\alpha$ , shift  $\beta$ , and base  $\gamma$ .

#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  $\alpha$ , shift  $\beta$ , and base  $\gamma$ .

## 5.39.2 Constructor & Destructor Documentation

## 5.39.2.1 ExpLayer()

## **Parameters**

param

provides  $ExpParameter\ exp\_param,\ with\ ExpLayer\ options:$ 

- scale (optional, default 1) the scale  $\alpha$
- shift (optional, default 0) the shift  $\beta$
- base (optional, default -1 for a value of  $e \approx 2.718$ ) the base  $\gamma$

## 5.39.3 Member Function Documentation

## 5.39.3.1 Backward\_cpu()

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 $\frac{\partial E}{\partial x} = \frac{\partial E}{\partial y} y \alpha \log_e(gamma) \text{ if propagate\_down[0]}$

Implements caffe::Layer < Dtype >.

## 5.39.3.2 Forward\_cpu()

## **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()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, 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:

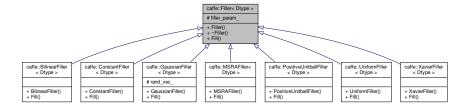
- · include/caffe/layers/exp layer.hpp
- src/caffe/layers/exp\_layer.cpp

## 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 >:

# caffe::Filler< Dtype > # filler\_param\_ + Filler() + ~Filler() + Fill()

## **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

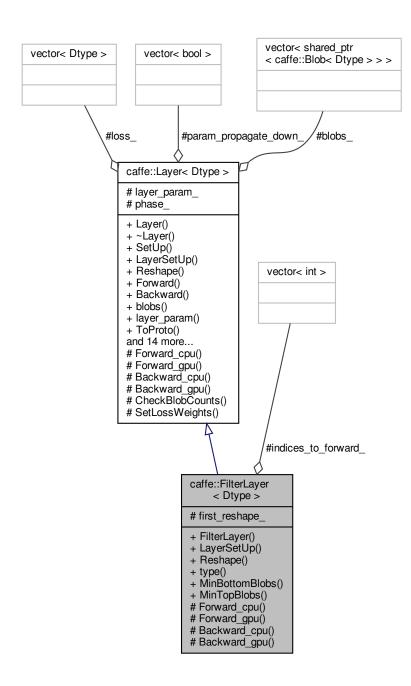
Takes two+ Blobs, interprets last Blob as a selector and filter remaining Blobs accordingly with selector data (0 means that the corresponding item has to be filtered, non-zero means that corresponding item needs to stay).

```
#include <filter_layer.hpp>
```

Inheritance diagram for caffe::FilterLayer< Dtype >:

# caffe::Layer< Dtype > # layer\_param\_ # phase # blobs\_ # param\_propagate\_down\_ # loss + Layer() + ~Layer() + SetUp() + LayerSetUp() + Reshape() + Forward() + Backward() + blobs() + layer\_param() + ToProto() and 14 more... # Forward\_cpu() # Forward\_gpu() # Backward\_cpu() # Backward\_gpu() # CheckBlobCounts() # SetLossWeights() caffe::FilterLayer < Dtype > # first\_reshape\_ # indices\_to\_forward\_ + FilterLayer() + LayerSetUp() + Reshape() + type() + MinBottomBlobs() + MinTopBlobs() # Forward\_cpu() # Forward\_gpu() # Backward\_cpu() # Backward\_gpu()

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()

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()

## **Parameters**

bottom	input Blob vector (length 2+)
	1. $(N  imes C  imes H  imes W)$ the inputs to be filtered $x_1$
	2
	3. $(N  imes C  imes H  imes W)$ 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()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, 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()

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

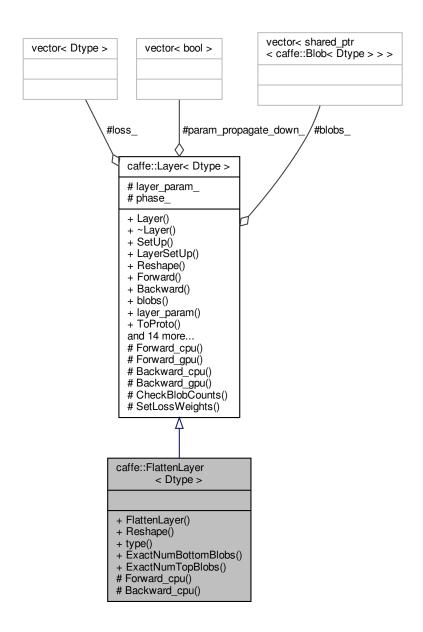
This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as

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.
<pre>#include <flatten_layer.hpp></flatten_layer.hpp></pre>

Inheritance diagram for caffe::FlattenLayer< Dtype >:

```
caffe::Layer< Dtype >
# layer_param_
# phase_
# blobs_
# param_propagate_down_
# loss
+ Layer()
+ ~Layer()
+ SetÚp()
+ LayerSetUp()
+ Reshape()
+ Forward()
+ Backward()
+ blobs()
+ layer_param()
+ ToProto()
and 14 more...
# Forward_cpu()
# Forward_gpu()
# Backward_cpu()
# Backward_gpu()
# CheckBlobCounts()
# SetLossWeights()
caffe::FlattenLayer
        < Dtype >
+ FlattenLayer()
+ Reshape()
+ type()
+ ExactNumBottomBlobs()
+ ExactNumTopBlobs()
# Forward_cpu()
# Backward_cpu()
```

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()

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()

## **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	
	1. (1) × 0 11 W × 1 × 1) the outputs – i.e., the (virtually) copied, hatteried 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:

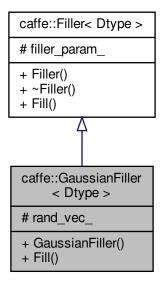
- include/caffe/layers/flatten\_layer.hpp
- src/caffe/layers/flatten layer.cpp

## 5.43 caffe::GaussianFiller < Dtype > Class Template Reference

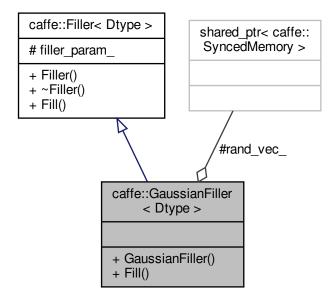
Fills a Blob with Gaussian-distributed values x = a.

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

Inheritance diagram for caffe::GaussianFiller < Dtype >:



 $\label{lem:collaboration} \mbox{Collaboration diagram for caffe::} \mbox{GaussianFiller} < \mbox{Dtype} > :$ 



## **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

 $\label{template} \mbox{template}{<}\mbox{typename Dtype}{>} \\ \mbox{class caffe::GaussianFiller}{<}\mbox{ Dtype}{>}$ 

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::RNG::Generator Class Reference

Collaboration diagram for caffe::Caffe::RNG::Generator:

caffe::Caffe::RNG:: Generator

- + Generator()
- + Generator()
- + rng()

## **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

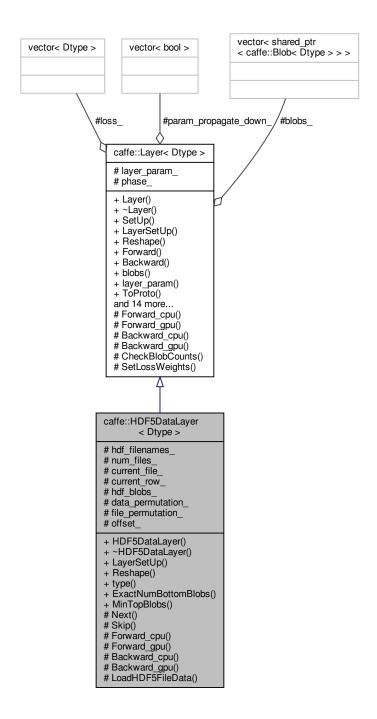
Provides data to the Net from HDF5 files.

#include <hdf5\_data\_layer.hpp>

Inheritance diagram for caffe::HDF5DataLayer< Dtype >:

```
caffe::Layer< Dtype >
# layer_param_
# phase_
# blobs
# param_propagate_down_
# loss
+ Layer()
+ ~Layer()
+ SetUp()
+ LayerSetUp()
+ Reshape()
+ Forward()
+ Backward()
+ blobs()
+ layer_param()
+ ToProto()
and 14 more...
# Forward_cpu()
# Forward_gpu()
# Backward_cpu()
# Backward_gpu()
# CheckBlobCounts()
# Set less Weights()
# SetLossWeights()
caffe::HDF5DataLayer
            < Dtype >
# hdf_filenames_
# num_files_
# current_file_
# current_row_
# hdf_blobs_
# data_permutation_
# file_permutation_
# offset
+ HDF5DataLayer()
+ ~HDF5DataLayer()
+ LayerSetUp()
+ Reshape()
+ type()
+ type()
+ ExactNumBottomBlobs()
+ MinTopBlobs()
# Next()
# Skip()
# Skip()
# Forward_cpu()
# Forward_gpu()
# Backward_cpu()
# Backward_gpu()
# LoadHDF5FileData()
```

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

```
\label{template} \mbox{template}{<} \mbox{typename Dtype}{>} \\ \mbox{class caffe}{::HDF5DataLayer}{<} \mbox{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()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, 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()

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

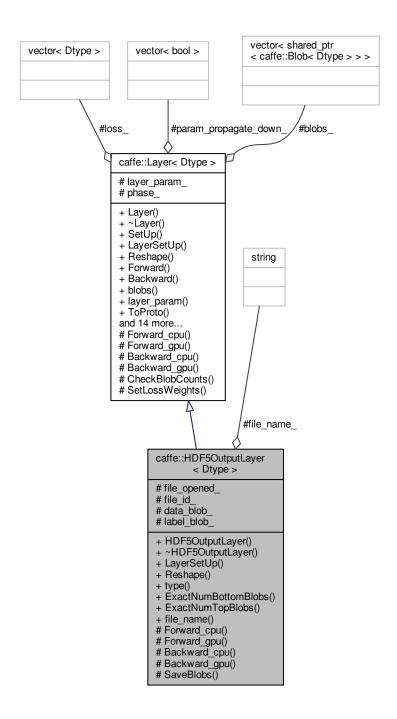
Write blobs to disk as HDF5 files.

#include <hdf5\_output\_layer.hpp>

Inheritance diagram for caffe::HDF5OutputLayer< Dtype >:

# caffe::Layer< Dtype > # layer\_param\_ # phase # blobs\_ # param\_propagate\_down\_ # İoss + Layer() + ~Layer() + SetUp() + LayerSetUp() + Reshape() + Forward() + Backward() + blobs() + layer\_param() + ToProto() and 14 more... # Forward\_cpu() # Forward\_gpu() # Backward\_cpu() # Backward\_gpu() # CheckBlobCounts() # SetLossWeights() caffe::HDF5OutputLayer < Dtype > # file\_opened\_ # file\_name\_ # file\_id # data\_blob\_ # label\_blob\_ + HDF5OutputLayer() + ~HDF5OutputLayer() + LayerSetUp() + Reshape() + type() + ExactNumBottomBlobs() + ExactNumTopBlobs() + file\_name() # Forward\_gpu() # Backward\_cpu() # Backward\_gpu() # SaveBlobs()

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()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

### 5.46.2.4 Reshape()

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:

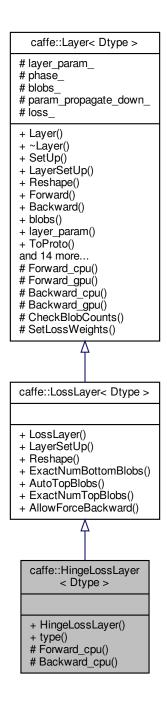
• include/caffe/layers/hdf5\_output\_layer.hpp

## 5.47 caffe::HingeLossLayer < Dtype > Class Template Reference

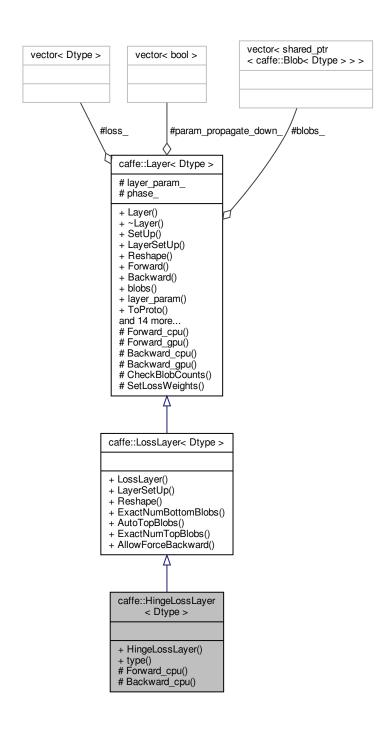
Computes the hinge loss for a one-of-many classification task.

#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 $X^TW$ 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: $E = \frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} [\max(0, 1 - \delta\{l_n = k\}t_{nk})]^p$ , for the $L^p$ norm (defaults to $p = 1$ , the L1 norm; L2 norm, as in L2-SVM, is also available), and $\delta\{\text{condition}\} = \begin{cases} 1 & \text{if condition} \\ -1 & \text{otherwise} \end{cases}$

In an SVM,  $t \in \mathcal{R}^{N \times K}$  is the result of taking the inner product  $X^TW$  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()

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* $\lambda$ , as $\lambda$ is the coefficient of this layer's output $\ell_i$ 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. $(N  imes C  imes H  imes W)$ the predictions $t$ ; Backward computes diff $rac{\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()

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 $X^TW$ 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: $E = \frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} [\max(0, 1 - \delta\{l_n = k\}t_{nk})]^p$ , for the $L^p$ norm (defaults to $p = 1$ , the L1 norm; L2 norm, as in L2-SVM, is also available), and $\delta\{\text{condition}\} = \begin{cases} 1 & \text{if condition} \\ -1 & \text{otherwise} \end{cases}$

In an SVM,  $t \in \mathcal{R}^{N \times K}$  is the result of taking the inner product  $X^TW$  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

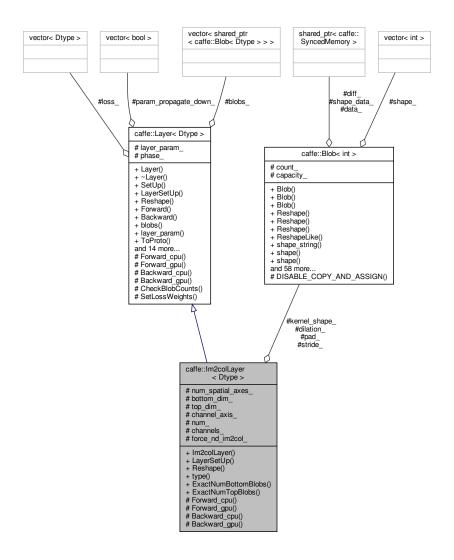
A helper for image operations that rearranges image regions into column vectors. Used by ConvolutionLayer to perform convolution by matrix multiplication.

#include <im2col\_layer.hpp>

Inheritance diagram for caffe::Im2colLayer< Dtype >:

# caffe::Layer< Dtype > # layer\_param\_ # phase\_ # blobs # param\_propagate\_down\_ # loss\_ + Layer() + ~Layer() + SetUp() + LayerSetUp() + Reshape() + Forward() + Backward() + blobs() + layer\_param() + ToProto() and 14 more... # Forward\_cpu() # Forward\_gpu() # Backward\_cpu() # Backward\_gpu() # CheckBlobCounts() # SetLossWeights() caffe::Im2colLayer < Dtype > # kernel\_shape\_ # stride\_ # pad\_ # dilation\_ # num\_spatial\_axes\_ # bottom\_dim\_ # top\_dim\_ # channel\_axis\_ # num\_ # channels\_ # force\_nd\_im2col + Im2colLayer() + LayerSetÚp() + Reshape() + type() + type() + ExactNumBottomBlobs() + ExactNumTopBlobs() # Forward\_cpu() # Forward\_gpu() # Backward\_cpu() # Backward\_gpu()

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()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

### 5.48.2.4 Reshape()

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

#### **Parameters**

bo	ttom	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:

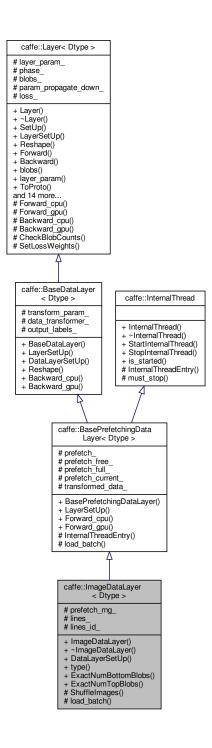
- include/caffe/layers/im2col\_layer.hpp
- · src/caffe/layers/im2col\_layer.cpp

# 5.49 caffe::ImageDataLayer < Dtype > Class Template Reference

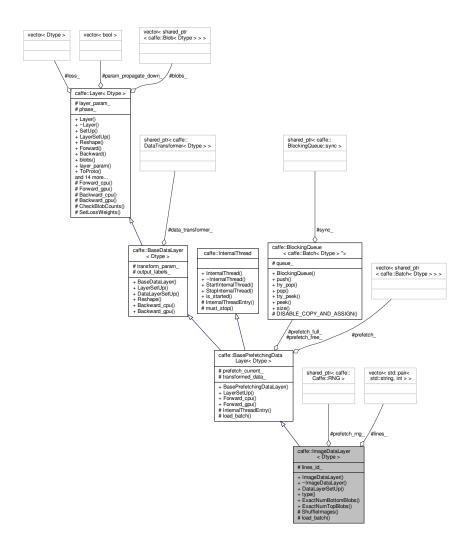
Provides data to the Net from image files.

```
#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 Shufflelmages ()
- 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:

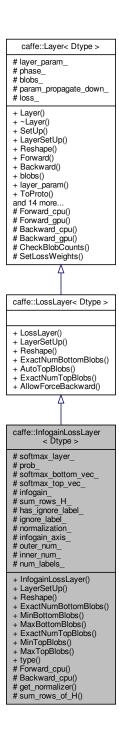
include/caffe/layers/image\_data\_layer.hpp

# 5.50 caffe::InfogainLossLayer < Dtype > Class Template Reference

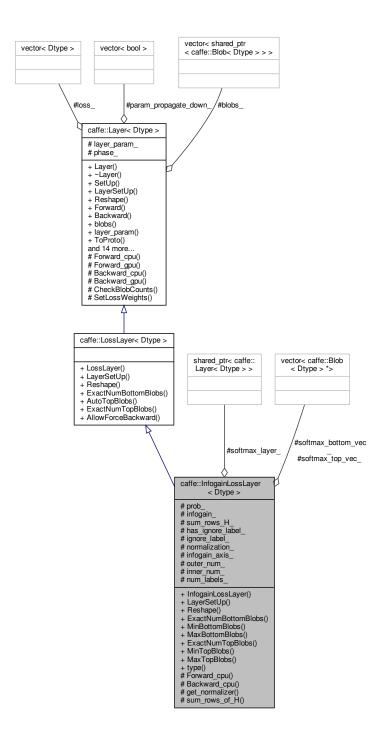
A generalization of SoftmaxWithLossLayer that takes an "information gain" (infogain) matrix specifying the "value" of all label pairs.

```
#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. $(1 \times 1 \times K \times K)$ ( <b>optional</b> ) 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:
	$E = \frac{-1}{N} \sum_{n=1}^{N} H_{l_n} \log(\hat{p}_n) = \frac{-1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} H_{l_n,k} \log(\hat{p}_{n,k}), \text{ where } H_{l_n} \text{ denotes row } l_n \text{ of } H.$

### 5.50.2 Member Function Documentation

### 5.50.2.1 Backward\_cpu()

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* $\lambda$ , as $\lambda$ is the coefficient of this layer's output $\ell_i$ 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. $(1 \times 1 \times K \times K)$ ( <b>optional</b> ) 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()

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. $(1 \times 1 \times K \times K)$ ( <b>optional</b> ) 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:
	$E = \frac{-1}{N} \sum_{n=1}^{N} H_{l_n} \log(\hat{p}_n) = \frac{-1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} H_{l_n,k} \log(\hat{p}_{n,k}), \text{ where } H_{l_n} \text{ denotes row } l_n \text{ of } H.$

Implements caffe::Layer< Dtype >.

### 5.50.2.5 get\_normalizer()

Read the normalization mode parameter and compute the normalizer based on the blob size. If normalization\_\circ} 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()

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()

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

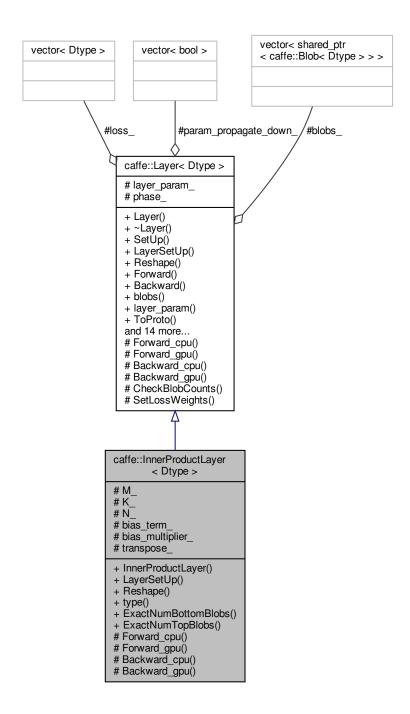
Also known as a "fully-connected" layer, computes an inner product with a set of learned weights, and (optionally) adds biases.

```
#include <inner_product_layer.hpp>
```

Inheritance diagram for caffe::InnerProductLayer< Dtype >:

# caffe::Layer< Dtype > # layer\_param\_ # phase # blobs\_ # param\_propagate\_down\_ # loss\_ + Layer() + ~Layer() + SetÚp() + LayerSetUp() + Reshape() + Forward() + Backward() + blobs() + layer\_param() + ToProto() and 14 more... # Forward\_cpu() # Forward\_gpu() # Backward\_cpu() # Backward\_gpu() # CheckBlobCounts() # SetLossWeights() caffe::InnerProductLayer < Dtype > # M\_ # K\_ # N\_ # bias\_term\_ # bias\_multiplier\_ # transpose\_ + InnerProductLayer() + LayerSetUp() + Reshape() + type() + ExactNumBottomBlobs() + ExactNumTopBlobs() # Forward\_cpu() # Forward\_gpu() # Backward\_cpu() # Backward\_gpu()

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()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

### 5.51.2.4 Reshape()

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

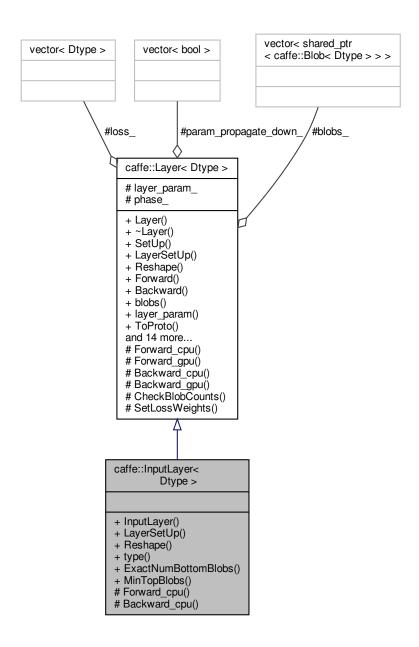
Provides data to the Net by assigning tops directly.

```
#include <input_layer.hpp>
```

Inheritance diagram for caffe::InputLayer< Dtype >:

# caffe::Layer< Dtype > # layer param # phase\_ # blobs\_ # param\_propagate\_down\_ # loss\_ + Layer() + ~Layer() + SetUp() + LayerSetUp() + Reshape() + Forward() + Backward() + blobs() + layer\_param() + ToProto() and 14 more... # Forward\_cpu() # Forward\_gpu() # Backward\_cpu() # Backward\_gpu() # CheckBlobCounts() # SetLossWeights() caffe::InputLayer< Dtype > + InputLayer() + LayerSetUp() + Reshape() + type() + ExactNumBottomBlobs() + MinTopBlobs() # Forward\_cpu() # Backward\_cpu()

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()

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

#### **Parameters**

b	ottom	the preshaped input blobs, whose data fields store the input data for this layer
to	р	the allocated but unshaped output blobs

This method should do one-time layer specific setup. This includes reading and processing relevent parameters from the <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, 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()

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 >.

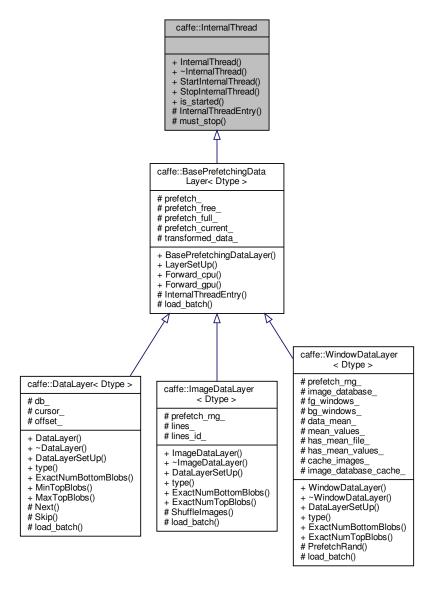
The documentation for this class was generated from the following files:

- 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:

# caffe::InternalThread

- + InternalThread()
- + ~InternalThread()
- + StartInternalThread()
- + StopInternalThread()
- + is started()
- # InternalThreadEntry()
- # must\_stop()

# **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:

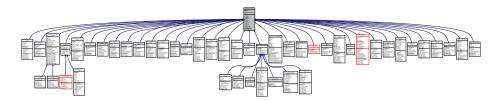
- include/caffe/internal\_thread.hpp
- src/caffe/internal\_thread.cpp

# 5.54 caffe::Layer < Dtype > Class Template Reference

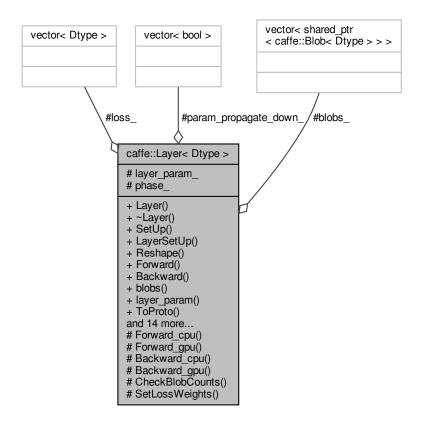
An interface for the units of computation which can be composed into a Net.

```
#include <layer.hpp>
```

Inheritance diagram for caffe::Layer< Dtype >:



Collaboration diagram for caffe::Layer< Dtype >:



## **Public Member Functions**

- Layer (const LayerParameter &param)
- $\bullet \ \ \mathsf{void} \ \mathsf{SetUp} \ (\mathsf{const} \ \mathsf{vector} < \mathsf{Blob} < \mathsf{Dtype} > *> \& \mathsf{bottom}, \ \mathsf{const} \ \mathsf{vector} < \mathsf{Blob} < \ \mathsf{Dtype} > *> \& \mathsf{top}) \\$

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.
- $\bullet \ \, {\rm Dtype} \ \, {\rm Forward} \ \, ({\rm const} \ \, {\rm vector} < \, {\rm Blob} < \, {\rm Dtype} > *> \, \& {\rm bottom}, \, {\rm const} \ \, {\rm vector} < \, {\rm Blob} < \, {\rm Dtype} > *> \, \& {\rm top})$

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()

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()

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()

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()

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::BatchReindexLayer< Dtype >, caffe::BatchReindexLayer< Dtype >, caffe::SPPLayer< Dtype >, caffe::MemoryData  $\leftarrow$  Layer< Dtype >, caffe::PoolingLayer< Dtype >, caffe::SpPLayer< Dtype >, caffe::MVNLayer< Dtype >, caffe::Dutype >, 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::SoftmaxWith  $\leftarrow$  LossLayer< Dtype >, caffe::BatchNormLayer< Dtype >, caffe::ArgMaxLayer< Dtype >, caffe::RecurrentLayer< Dtype >, caffe::LossLayer< Dtype >, caffe::BatchNormLayer< Dtype >, caffe::HDF5OutputLayer< Dtype >, caffe:: $\leftarrow$  WindowDataLayer< Dtype >, caffe::BiasLayer< Dtype >, caffe::BiasLayer< Dtype >, caffe::LRNLayer< Dtype >, caffe::ImageDataLayer< Dtype >, caffe::CropLayer< Dtype >, caffe::FlattenLayer< Dtype >, caffe::Bmbed  $\leftarrow$  Layer< Dtype >, caffe::Im2colLayer< Dtype >, caffe::ReductionLayer< Dtype >, caffe::BatchReindexLayer< Dtype >, caffe::BatchReindexLayer< Dtype >, caffe::BetwiseLayer< Dtype >, caffe::ParameterLayer< Dtype >, caffe::ReshapeLayer< Dtype >, caffe::SPPLayer< Dtype >, caffe::MemoryDataLayer< Dtype >, caffe::Concat  $\leftarrow$  Layer< Dtype >, caffe::MeuronLayer< Dtype >, caffe::SoftmaxLayer< Dtype >, caffe::SilenceLayer< Dtype >, caffe::TileLayer< Dtype >.

#### 5.54.3.8 Forward()

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()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented in caffe::BasePrefetchingDataLayer< Dtype >, caffe::SoftmaxWithLossLayer< Dtype >, caffe  $\leftarrow$  ::InfogainLossLayer< Dtype >, caffe::SigmoidCrossEntropyLossLayer< Dtype >, caffe::BatchNormLayer< Dtype >, caffe::ContrastiveLossLayer< Dtype >, caffe::ArgMaxLayer< Dtype >, caffe::DropoutLayer< Dtype >, caffe  $\leftarrow$  ::PReLULayer< Dtype >, caffe::SwishLayer< Dtype >, caffe::ExpLayer< Dtype >, caffe::LogLayer< Dtype >, caffe::AccuracyLayer< Dtype >, caffe::PowerLayer< Dtype >, caffe::RecurrentLayer< Dtype >, caffe::Scale  $\leftarrow$  Layer< Dtype >, caffe::AbsValLayer< Dtype >, caffe::HDF5OutputLayer< Dtype >, caffe::ThresholdLayer< Dtype >, caffe::HDF5DataLayer< Dtype >, caffe::BaseDataLayer< Dtype >, caffe::LossLayer< Dtype >, caffe::LeosLayer< Dtype >, caffe::BiasLayer< Dtype >, caffe::CropLayer< Dtype >, caffe::EmbedLayer< Dtype >, caffe::Im2colLayer< Dtype >, caffe::BaseDataLayer< Dtype >, caffe::DummyDataLayer< Dtype >, caffe::HDF5DataLayer< Dtype >, caffe::DummyDataLayer< Dtype >, caffe::Hothlayer< Dtype >, caffe::DummyDataLayer< Dtype >, caffe::DumyDataLayer< Dtype >, caffe::BaseCatellayer< 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::Accuracy  $\leftarrow$  Layer< 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::BiasLayer< Dtype >, caffe::Base  $\leftarrow$  ConvolutionLayer< 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::Accuracy  $\leftarrow$  Layer< Dtype >, caffe::HDF5DataLayer< Dtype >, caffe::DummyDataLayer< Dtype >, caffe::InputLayer< Dtype >, caffe::FilterLayer< Dtype >, caffe::Base  $\leftarrow$  ConvolutionLayer< Dtype >, caffe::SplitLayer< Dtype >, and caffe::DataLayer< Dtype >.

## 5.54.3.14 param\_propagate\_down()

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()

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.

 $\label{loss_layer} Implemented in caffe::LSTMUnitLayer < Dtype >, caffe::SoftmaxWithLossLayer < Dtype >, caffe::InfogainLoss \leftarrow Layer < Dtype >, caffe::SigmoidCrossEntropyLossLayer < Dtype >, caffe::MultinomialLogisticLossLayer < Dtype >, caffe::BatchNormLayer < Dtype >, caffe::EuclideanLossLayer < Dtype >, caffe::ArgMaxLayer < Dtype >, caffe::PythorLayer < Dtype >, caffe::PythorLayer < Dtype >, caffe::BaseDataLayer < Dtype >, caffe::HDF5OutputLayer < Dtype >, caffe::PythorLayer < Dtype >, caffe::RecurrentLayer < Dtype >, caffe::ScaleLayer < Dtype >, caffe::HDF5DataLayer < Dtype >, caffe::Loss \leftarrow Layer < Dtype >, caffe::LRNLayer < Dtype >, caffe::BiasLayer < Dtype >, caffe::CropLayer < Dtype >, caffe::Hm2colLayer < Dtype >, caffe::InputLayer < Dtype >, caffe::ParameterLayer < Dtype >, caffe::ReductionLayer < Dtype >, caffe::BatchReindexLayer < Dtype >, caffe::BltwiseLayer < Dtype >, caffe::BlterLayer < Dtype >, caffe::InnerProduct \leftarrow Layer < Dtype >, caffe::ReshapeLayer < Dtype >, caffe::SliceLayer < Dtype >, caffe::SPPLayer < Dtype >, caffe::BaseConvolutionLayer < Dtype >, caffe::Neuron \leftarrow Layer < Dtype >, caffe::SplitLayer < Dtype >, caffe::MVNLayer < Dtype >, caffe::SoftmaxLayer < Dtype >, caffe::SilenceLayer < Dtype >, caffe::TileLayer < Dtype >, caffe::TileLayer < Dtype >, caffe::SoftmaxLayer < Dtype >, caffe::TileLayer  

# 5.54.3.16 SetLossWeights()

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()

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**

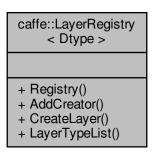
• LayerRegisterer (const string &type, shared\_ptr< Layer< Dtype > >(\*creator)(const LayerParameter &))

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 &)
- $\bullet \ \ \ type def \ std::map{<} \ string, \ Creator > \textbf{CreatorRegistry}$

# **Static Public Member Functions**

•	static	CreatorRed	istrv &	Registry	(	)
	Static	Or Calorrice	pioli y G	i icgisti y	١.	

- static void **AddCreator** (const string &type, Creator creator)
- static shared\_ptr< Layer< Dtype >> CreateLayer (const LayerParameter &param)
- static vector< string > LayerTypeList ()

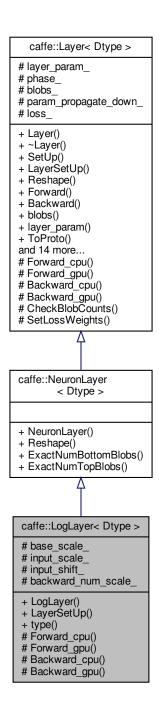
The documentation for this class was generated from the following file:

include/caffe/layer\_factory.hpp

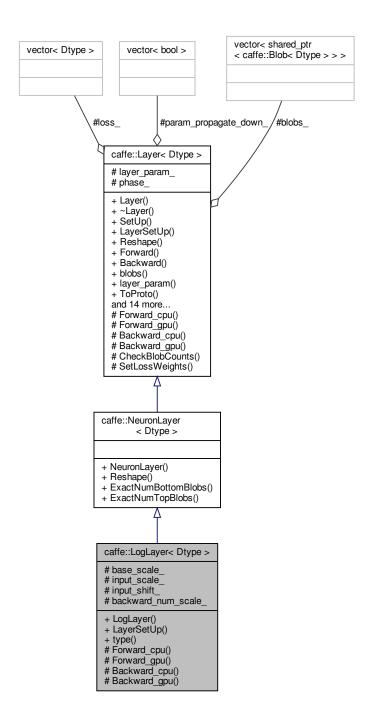
# 5.57 caffe::LogLayer < Dtype > Class Template Reference

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

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  $\alpha$ , shift  $\beta$ , and base  $\gamma$ .

# 5.57.2 Constructor & Destructor Documentation

## 5.57.2.1 LogLayer()

## **Parameters**

#### param

provides LogParameter log\_param, with LogLayer options:

- scale (optional, default 1) the scale  $\alpha$
- shift (**optional**, default 0) the shift  $\beta$
- base (optional, default -1 for a value of  $e \approx 2.718$ ) the base  $\gamma$

# 5.57.3 Member Function Documentation

# 5.57.3.1 Backward\_cpu()

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 $\frac{\partial E}{\partial x} = \frac{\partial E}{\partial y} y \alpha \log_e(gamma) \text{ if propagate\_down[0]}$	

Implements caffe::Layer < Dtype >.

# 5.57.3.2 Forward\_cpu()

## **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()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, 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:

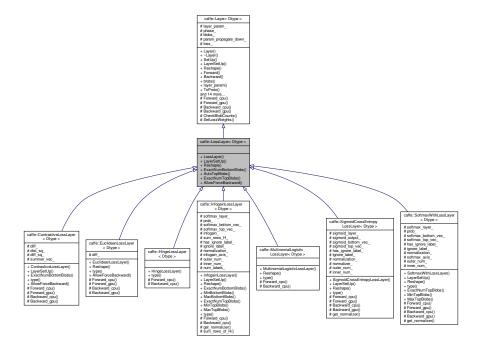
- include/caffe/layers/log\_layer.hpp
- src/caffe/layers/log layer.cpp

# 5.58 caffe::LossLayer < Dtype > Class Template Reference

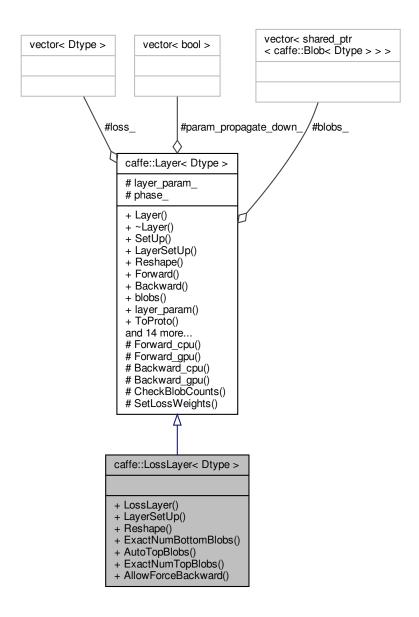
An interface for Layers that take two Blobs as input – usually (1) predictions and (2) ground-truth labels – and output a singleton Blob representing the loss.

```
#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 Loss← Layers, 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()

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()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, 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::Sigmoid $\leftarrow$  CrossEntropyLossLayer< Dtype >, and caffe::ContrastiveLossLayer< Dtype >.

## 5.58.2.5 Reshape()

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  $\leftarrow$  CrossEntropyLossLayer< Dtype >, caffe::MultinomialLogisticLossLayer< Dtype >, and caffe::EuclideanLoss  $\leftarrow$  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

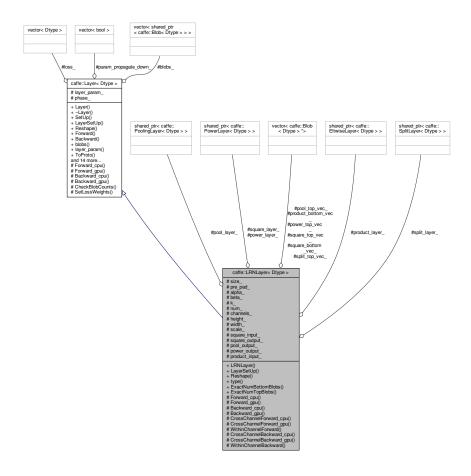
Normalize the input in a local region across or within feature maps.

#include <lrn\_layer.hpp>

Inheritance diagram for caffe::LRNLayer< Dtype >:

```
caffe::Layer< Dtype >
      # layer_param_
      # phase_
      # blobs
      # param_propagate_down_
      # loss_
     + Layer()
+ ~Layer()
+ SetUp()
+ LayerSetUp()
+ Reshape()
      + Forward()
      + Backward()
      + blobs()
      + layer_param()
+ ToProto()
      and 14 more..
      # Forward_cpu()
# Forward_gpu()
# Backward_cpu()
     # Backward_gpu()
# CheckBlobCounts()
# SetLossWeights()
      caffe::LRNLayer< Dtype >
 # size_
# pre_pad_
# alpha_
# beta_
# num_
# channels_
 # height_
 # width_
 # scale
and 16 more...
+ LRNLayer()
+ LayerSetUp()
+ Reshape()
+ Hesnape()
+ type()
+ ExactNumBottomBlobs()
+ ExactNumTopBlobs()
# Forward_cpu()
# Forward_dpu()
# Backward_cpu()
#Backward_cpu()
#Backward_gpu()
#CrossChannelForward_cpu()
#CrossChannelForward_gpu()
#WithinChannelForward()
#CrossChannelBackward_cpu()
#CrossChannelBackward_gpu()
#WithinChannelBackward()
```

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</li>
   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</li>
   Dtype > \* > square\_top\_vec\_
- shared\_ptr< PoolingLayer< Dtype > > pool\_layer\_
- Blob < Dtype > pool\_output\_
- vector< Blob</li>
   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()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

#### 5.59.2.4 Reshape()

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.

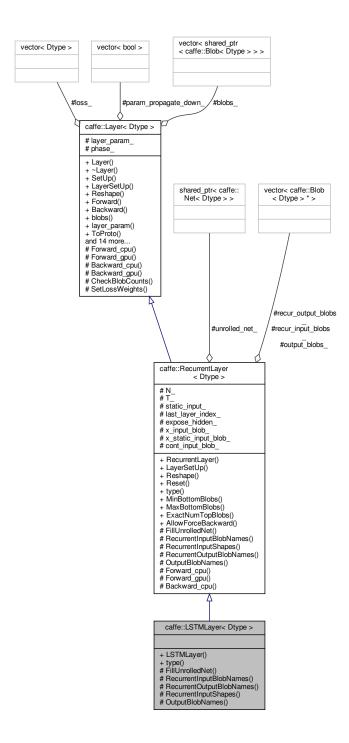
```
#include <1stm_layer.hpp>
```

Inheritance diagram for caffe::LSTMLayer< Dtype >:

```
caffe::Layer< Dtype >
           # layer_param_
# phase_
# blobs_
           # param_propagate_down_
# loss_
          + Layer()
+ ~Layer()
+ SetUp()
+ LayerSetUp()
+ Reshape()
          + Reshape()
+ Forward()
+ Backward()
+ blobs()
+ layer_param()
+ ToProto()
and 14 more...
# Forward_cpu()
# Forward_dpu()
# Backward_dpu()
# Backward_dpu()
# CheckBlobCounts()
# SetLossWeightts()
  caffe::RecurrentLayer < Dtype >
# unrolled_net_
# N_
# T_
# static_input
# last_layer_index_
# expose_hidden_
# recur_input_blobs_
# output_blobs_
# x_ input_blob_
# x_ static_input_blob_
# cour_output_blob_
# cour_output_blob_
# x_ input_blob_
# cour_input_blob_
# cont_input_blob_

+ RecurrentLayer()
+ LayerSetUp()
+ Reshape()
+ Reset()
+ type()
+ MinBottomBlobs()
+ MaxBottomBlobs()
+ ExactNumTopBlobs()
+ AllowForceBackward()
# iiiIUnrolledNet()
# RecurrentInputBlobNames()
# RecurrentInputBlobNames()
# RecurrentOutputBlobNames()
# OutputBlobNames()
# Forward_cpu()
   # Forward_cpu()
# Forward_gpu()
# Backward_cpu()
         caffe::LSTMLayer< Dtype >
    + LSTMLayer()
  + LS IMLayer()
+ type()
# FillUnrolledNet()
# RecurrentInputBlobNames()
# RecurrentInputBlobNames()
# RecurrentInputBhapes()
# OutputBlobNames()
```

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

 $\label{lem:condition} \begin{tabular}{ll} template < typename \ Dtype > \\ class \ caffe:: LSTMLayer < Dtype > \\ \end{tabular}$ 

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:

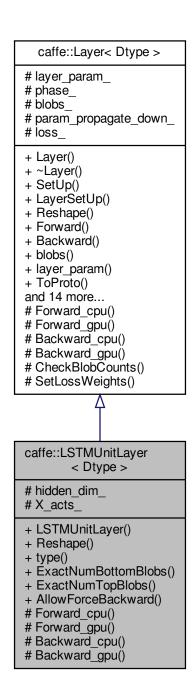
- include/caffe/layers/lstm\_layer.hpp
- src/caffe/layers/lstm\_layer.cpp

# 5.61 caffe::LSTMUnitLayer < Dtype > Class Template Reference

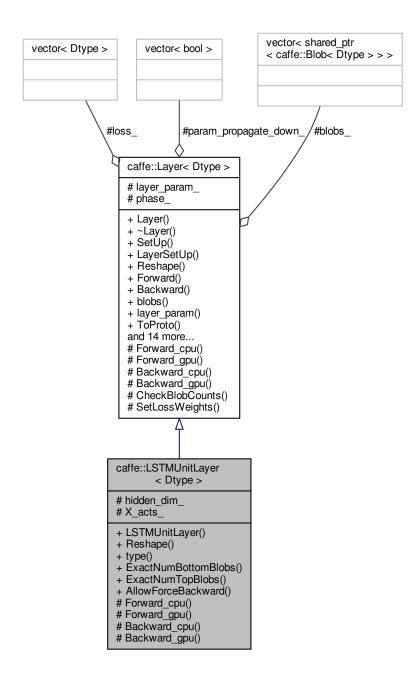
A helper for LSTMLayer: computes a single timestep of the non-linearity of the LSTM, producing the updated cell and hidden states.

```
#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()

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).

 $\label{lem:lemented:lemented:lemented} \mbox{Reimplemented from caffe::Layer} < \mbox{Dtype} >.$ 

#### 5.61.2.2 Backward\_cpu()

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 $ \begin{tabular}{ll} 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 $h_t$ \\ \end{tabular} $
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 $\delta_t$ 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()

#### **Parameters**

bottom	input Blob vector (length 3)
	1. $(1  imes N  imes D)$ 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. $(1  imes N)$ the sequence continuation indicators $\delta_t$
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()

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:

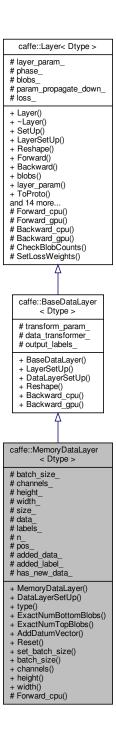
- include/caffe/layers/lstm layer.hpp
- src/caffe/layers/lstm\_unit\_layer.cpp

## 5.62 caffe::MemoryDataLayer < Dtype > Class Template Reference

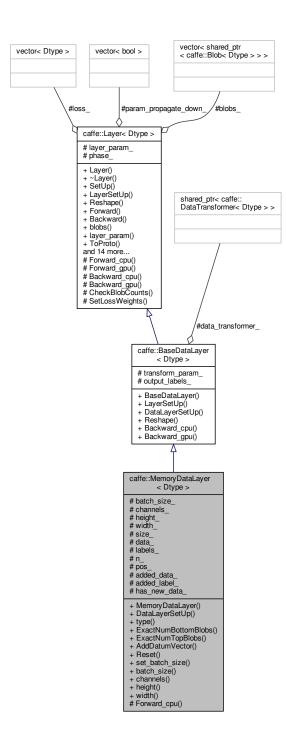
Provides data to the Net from memory.

```
#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:

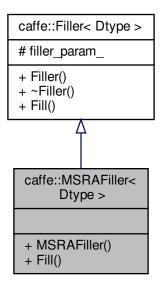
- include/caffe/layers/memory\_data\_layer.hpp
- src/caffe/layers/memory\_data\_layer.cpp

## 5.63 caffe::MSRAFiller < Dtype > Class Template Reference

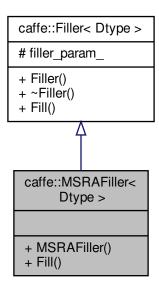
Fills a Blob with values  $x \sim N(0, \sigma^2)$  where  $\sigma^2$  is set inversely proportional to number of incoming nodes, outgoing nodes, or their average.

```
#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	<b>Members</b>
------------	-----------	----------------

### 5.63.1 Detailed Description

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

Fills a Blob with values  $x \sim N(0, \sigma^2)$  where  $\sigma^2$  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:

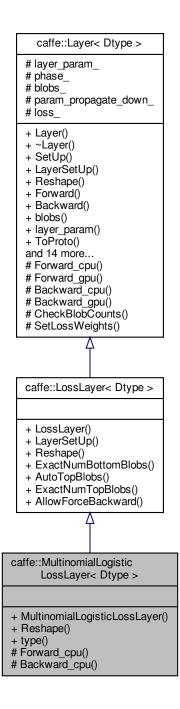
· include/caffe/filler.hpp

## 5.64 caffe::MultinomialLogisticLossLayer < Dtype > Class Template Reference

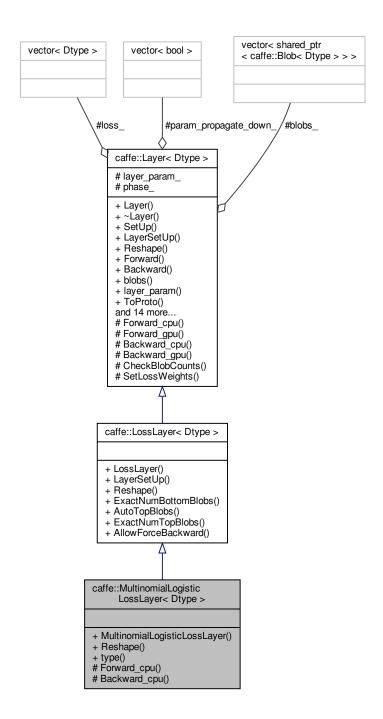
Computes the multinomial logistic loss for a one-of-many classification task, directly taking a predicted probability distribution as input.

#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: $\forall n \sum_{k=1}^K \hat{p}_{nk} = 1$ .
	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: $E = \frac{-1}{N} \sum_{n=1}^{N} \log(\hat{p}_{n,l_n})$

### 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 $ \begin{tabular}{ll} 1. & (1\times 1\times 1\times 1) \begin{tabular}{ll} This Blob's diff will simply contain the loss_weight* $\lambda$, as $\lambda$ is the coefficient of this layer's output $\ell_i$ 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()

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: $\forall n \sum_{k=1}^K \hat{p}_{nk} = 1$ .
	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)  Generated by Doxygen
	1. $(1 \times 1 \times 1 \times 1)$ the computed multinomial logistic loss: $E = \frac{-1}{N} \sum_{n=1}^{N} \log(\hat{p}_{n,l_n})$

Implements caffe::Layer< Dtype >.

#### 5.64.2.3 Reshape()

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

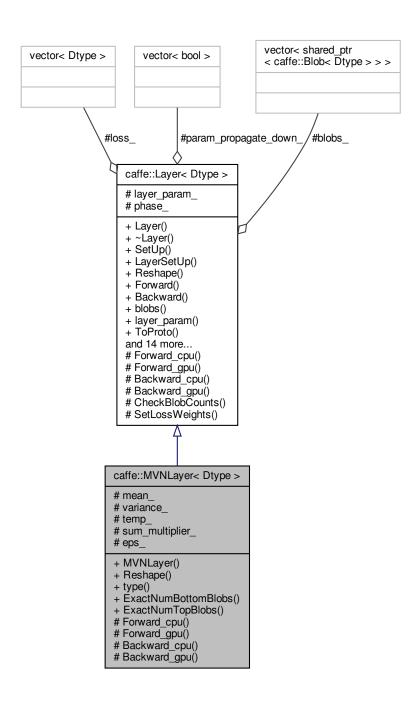
Normalizes the input to have 0-mean and/or unit (1) variance.

```
#include <mvn_layer.hpp>
```

Inheritance diagram for caffe::MVNLayer< Dtype >:

# caffe::Layer< Dtype > # layer\_param\_ # phase\_ # blobs\_ # param\_propagate\_down\_ # loss\_ + Layer() + ~Layer() + SetUp() + LayerSetUp() + Reshape() + Forward() + Backward() + blobs() + layer\_param() + ToProto() and 14 more... # Forward\_cpu() # Forward\_gpu() # Backward\_cpu() # Backward\_gpu() # CheckBlobCounts() # SetLossWeights() caffe::MVNLayer< Dtype > # mean\_ # variance\_ # temp\_ # sum\_multiplier\_ # eps\_ + MVNLayer() + Reshape() + type() + ExactNumBottomBlobs() + ExactNumTopBlobs() # Forward\_cpu() # Forward\_gpu() # Backward\_cpu() # Backward\_gpu()

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()

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.

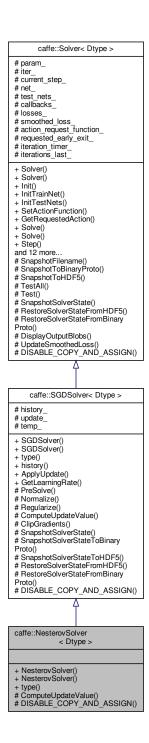
Implements caffe::Layer< Dtype >.

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

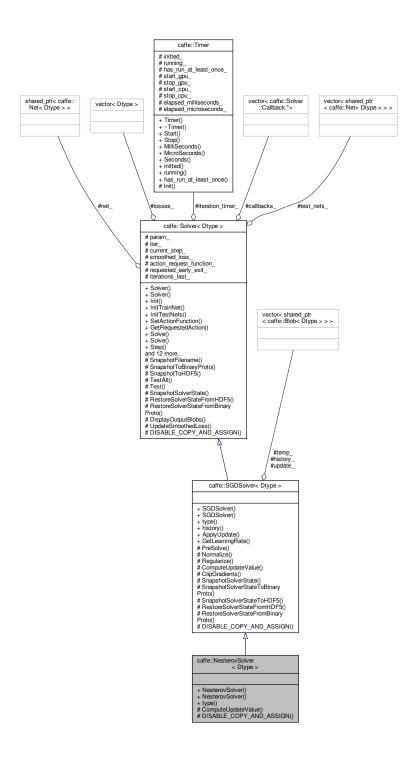
- include/caffe/layers/mvn\_layer.hpp
- src/caffe/layers/mvn\_layer.cpp

## 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<  ${\tt Blob}{\tt <Dtype}>>> {\tt \&params}$  () const

returns the parameters

const vector< Blob</li>
 Dtype > \* > & learnable\_params () const

- const vector< float > & params\_Ir () const
  - returns the learnable parameter learning rate multipliers
- const vector< bool > & has\_params\_Ir () 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</li>
    Dtype > * > net_input_blobs_

    vector< Blob</li>
    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 Ir

     the learning rate multipliers for learnable_params_

    vector< bool > has params Ir_

    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()

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:

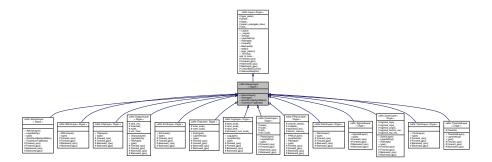
- · include/caffe/net.hpp
- src/caffe/net.cpp

## 5.68 caffe::NeuronLayer < Dtype > Class Template Reference

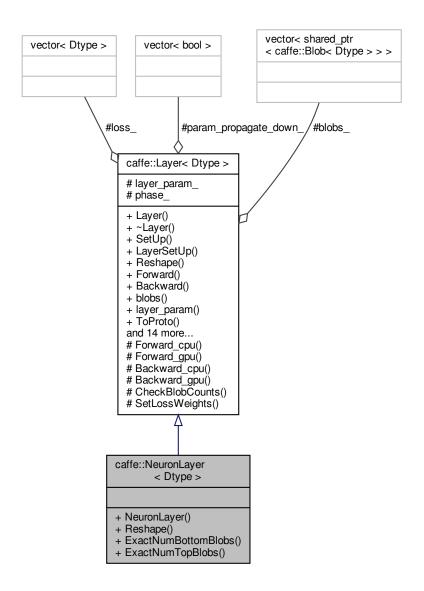
An interface for layers that take one blob as input (x) and produce one equally-sized blob as output (y), where each element of the output depends only on the corresponding input element.

```
#include <neuron_layer.hpp>
```

 $Inheritance\ diagram\ for\ caffe:: Neuron Layer < \ Dtype >:$ 



Collaboration diagram for caffe::NeuronLayer< Dtype >:



### **Public Member Functions**

- NeuronLayer (const LayerParameter &param)
- $\bullet \ \ \text{virtual void Reshape (const vector} < \ \ \text{Blob} < \ \ \text{Dtype} > *> \& \text{bottom, const vector} < \ \ \text{Blob} < \ \ \text{Dtype} > *> \& \text{top)}$

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()

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 >.

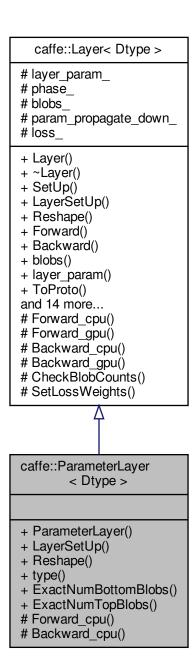
 $\label{lem:propoutLayer} Reimplemented in caffe:: PReLULayer < Dtype >, caffe:: DropoutLayer < Dtype >, and caffe:: SwishLayer < Dtype >.$ 

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

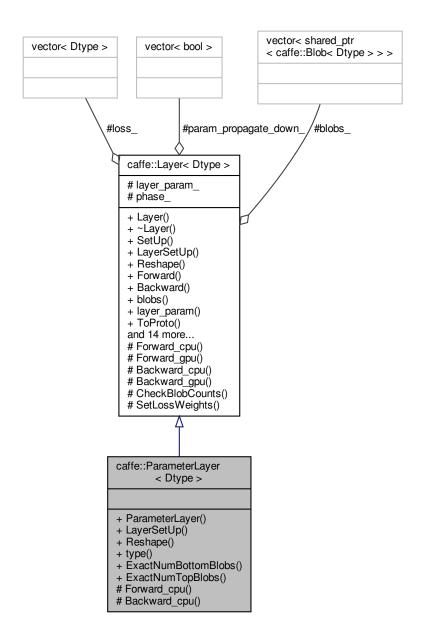
- include/caffe/layers/neuron\_layer.hpp
- src/caffe/layers/neuron\_layer.cpp

## 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()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

#### 5.69.1.4 Reshape()

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

#### **Parameters**

bott	om	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

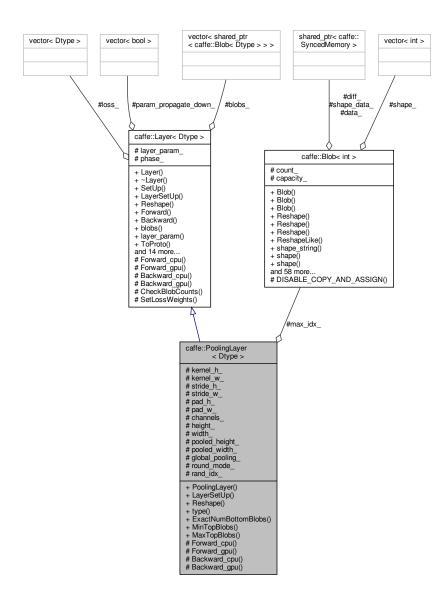
Pools the input image by taking the max, average, etc. within regions.

```
#include <pooling_layer.hpp>
```

Inheritance diagram for caffe::PoolingLayer< Dtype >:

```
caffe::Layer< Dtype >
 # layer_param_
 # phase_
# blobs
 # param_propagate_down_
 # loss_
+ Layer()
+ ~Layer()
+ SetUp()
+ LayerSetUp()
+ Reshape()
+ Forward()
+ Backward()
+ blobs()
+ layer_param()
+ ToProto()
and 14 more..
and 14 more...
# Forward_cpu()
# Forward_gpu()
# Backward_cpu()
# Backward_gpu()
# CheckBlobCounts()
# SetLossWeights()
caffe::PoolingLayer < Dtype >
# kernel_h_
# kernel_w_
# stride_h_
# stride_w_
# pad_h_
# pad_w_
# channels_
# height_
# width_
# pooled_height_
# pooled_width_
# global_pooling_
# round_mode_
 # rand_idx
 # max_idx
+ PoolingLayer()
+ LayerSetUp()
+ Reshape()
+ type()
+ ExactNumBottomBlobs()
+ MinTopBlobs()
+ MaxTopBlobs()
# Forward_opu()
# Forward_gpu()
# Backward_cpu()
 # Backward_gpu()
```

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()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, 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()

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:

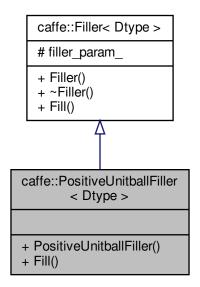
- · include/caffe/layers/pooling\_layer.hpp
- src/caffe/layers/pooling layer.cpp

#### 

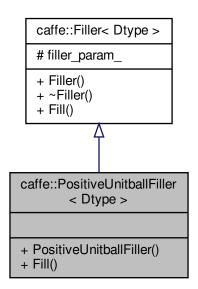
```
Fills a Blob with values x \in [0,1] such that \forall i \sum_j x_{ij} = 1.
```

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

 $Inheritance\ diagram\ for\ caffe:: Positive Unit ball Filler < \ Dtype >:$ 



 $\label{lem:collaboration} \mbox{Collaboration diagram for caffe::} \mbox{PositiveUnitballFiller} < \mbox{Dtype} > :$ 



### **Public Member Functions**

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

Additional	Inherited	<b>Members</b>
------------	-----------	----------------

## 5.71.1 Detailed Description

 $\label{template} \mbox{template} < \mbox{typename Dtype} > \\ \mbox{class caffe} :: \mbox{PositiveUnitballFiller} < \mbox{Dtype} > \\$ 

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:

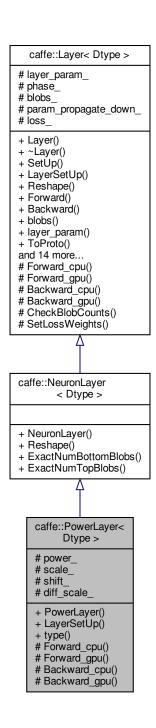
• include/caffe/filler.hpp

# 5.72 caffe::PowerLayer < Dtype > Class Template Reference

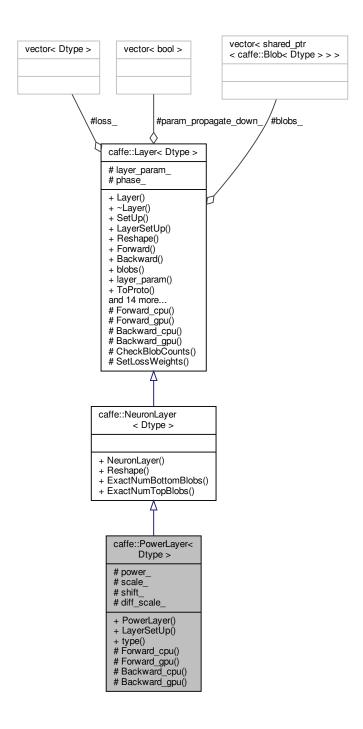
Computes  $y = (\alpha x + \beta)^{\gamma}$ , as specified by the scale  $\alpha$ , shift  $\beta$ , and power  $\gamma$ .

#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_
```

```
\gamma from layer_param_.power_param()
```

Dtype scale

```
\alpha from layer_param_.power_param()
```

Dtype shift

```
\beta 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  $\alpha$ , shift  $\beta$ , and power  $\gamma$ .

#### 5.72.2 Constructor & Destructor Documentation

#### 5.72.2.1 PowerLayer()

#### **Parameters**

```
\begin{array}{c} \textit{param} & \textit{provides PowerParameter power\_param, with PowerLayer} \\ & \textit{options:} \\ \\ & \bullet \;\; \textit{scale (optional, default 1) the scale } \alpha \\ \\ & \bullet \;\; \textit{shift (optional, default 0) the shift } \beta \\ \\ & \bullet \;\; \textit{power (optional, default 1) the power } \gamma \end{array}
```

#### 5.72.3 Member Function Documentation

## 5.72.3.1 Backward\_cpu()

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()

#### **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()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, 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:

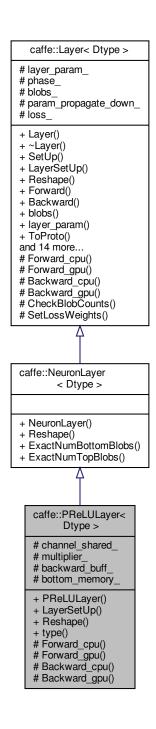
- include/caffe/layers/power\_layer.hpp
- src/caffe/layers/power\_layer.cpp

## 5.73 caffe::PReLULayer < Dtype > Class Template Reference

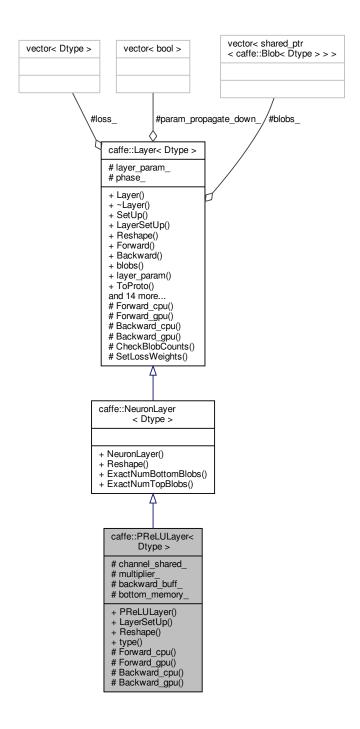
Parameterized Rectified Linear Unit non-linearity  $y_i = \max(0, x_i) + a_i \min(0, x_i)$ . The differences from ReLU $\leftarrow$  Layer are 1) negative slopes are learnable though backprop and 2) negative slopes can vary across channels. The number of axes of input blob should be greater than or equal 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 ReLU $\leftarrow$  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.

## 5.73.2 Constructor & Destructor Documentation

## 5.73.2.1 PReLULayer()

#### **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()

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 $ \text{1. } (N \times C \times) \text{ containing error gradients } \frac{\partial E}{\partial y} \text{ with respect to computed outputs } y $
propagate_down	see Layer::Backward.
bottom	input Blob vector (length 1) $ 1. \   (N \times C \times) \text{ the inputs } x; \text{For each channel } i, \text{ backward fills their diff with gradients} \\ \frac{\partial E}{\partial x_i} = \left\{ \begin{array}{ll} a_i \frac{\partial E}{\partial y_i} & \text{if } x_i \leq 0 \\ \frac{\partial E}{\partial y_i} & \text{if } x_i > 0 \end{array} \right. \text{ If param_propagate\_down_[0] is true, it fills the diff} \\ \text{with gradients } \frac{\partial E}{\partial a_i} = \left\{ \begin{array}{ll} \sum_{x_i} x_i \frac{\partial E}{\partial y_i} & \text{if } x_i \leq 0 \\ 0 & \text{if } x_i > 0 \end{array} \right $

Implements caffe::Layer< Dtype >.

## 5.73.3.2 Forward\_cpu()

#### **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()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

#### 5.73.3.4 Reshape()

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

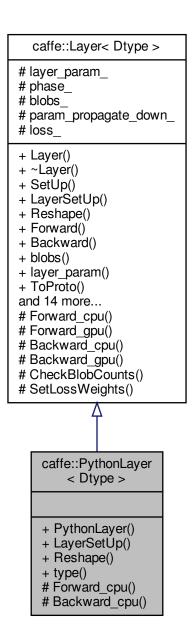
#### **Parameters**

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

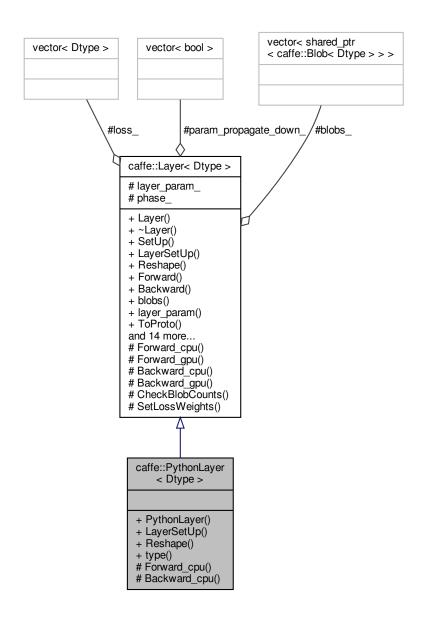
This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.
Reimplemented from caffe::NeuronLayer< Dtype >.
The documentation 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()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

## 5.74.1.2 Reshape()

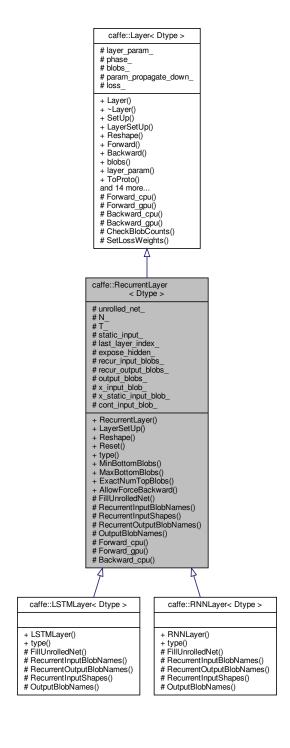
Adjust the shapes of top blobs and internal buffers to accommodate the shapes of the bottom blobs.

#### **Parameters**

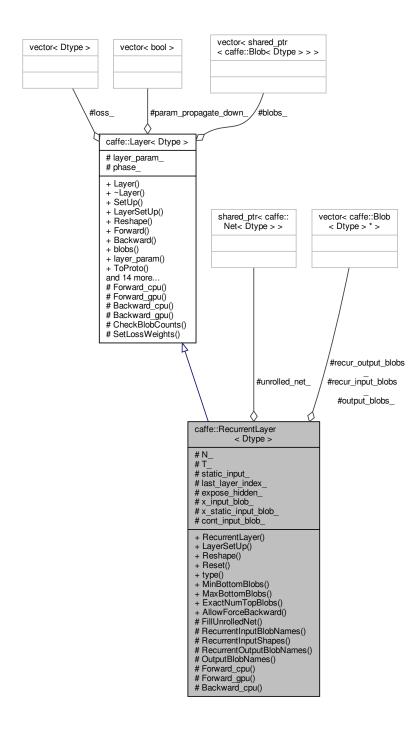
bottom	the input blobs, with the requested input shapes
top	the top blobs, which should be reshaped as needed

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.
<pre>#include <recurrent_layer.hpp></recurrent_layer.hpp></pre>

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 LSTM Layer 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 RNN← Layer 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**

 $\bullet \hspace{0.2cm} \mathsf{shared\_ptr} \! < \mathsf{Net} \! < \mathsf{Dtype} > \! > \! \mathsf{unrolled\_net\_}$ 

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()

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()

#### **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  $\delta$ . 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 t is a continuation from the previous timestep t-1, and the previous hidden state t0 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()

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()

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

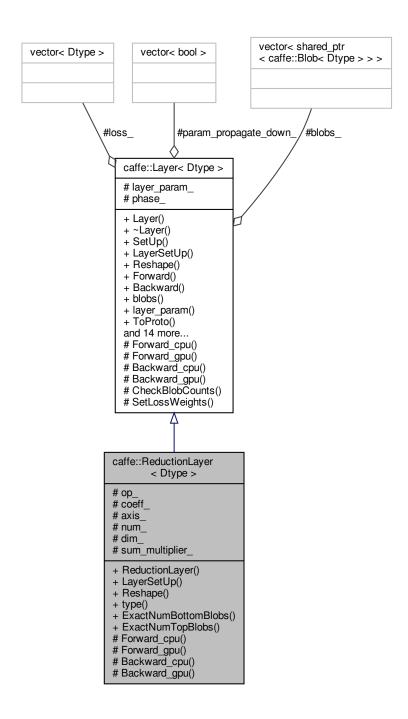
This method should reshape top blobs as needed according to the shapes of the bottom (input) blobs, as well as reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate the bottom blobs.

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.
<pre>#include <reduction_layer.hpp></reduction_layer.hpp></pre>

Inheritance diagram for caffe::ReductionLayer< Dtype >:

# caffe::Layer< Dtype > # layer\_param\_ # phase # blobs\_ # param\_propagate\_down\_ # loss\_ + Layer() + ~Layer() + SetÚp() + LayerSetUp() + Reshape() + Forward() + Backward() + blobs() + layer\_param() + ToProto() and 14 more... # Forward\_cpu() # Forward\_gpu() # Backward\_cpu() # Backward\_gpu() # CheckBlobCounts() # SetLossWeights() caffe::ReductionLayer < Dtype > # op\_ # coeff\_ # axis\_ # num\_ # dim\_ # sum\_multiplier\_ + ReductionLayer() + LayerSetUp() + Reshape() + type() + ExactNumBottomBlobs() + ExactNumTopBlobs() # Forward\_cpu() # Forward\_gpu() # Backward\_cpu() # Backward\_gpu()

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

```
\label{template} \mbox{template}{<} \mbox{typename Dtype}{>} \\ \mbox{class caffe::ReductionLayer}{<} \mbox{ 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()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

#### 5.76.2.4 Reshape()

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:

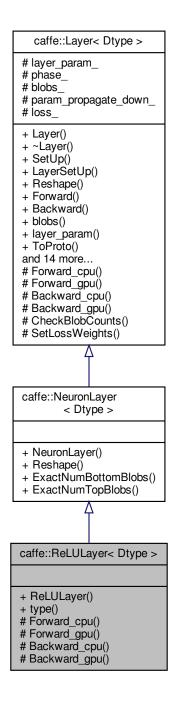
- include/caffe/layers/reduction\_layer.hpp
- src/caffe/layers/reduction layer.cpp

## 5.77 caffe::ReLULayer < Dtype > Class Template Reference

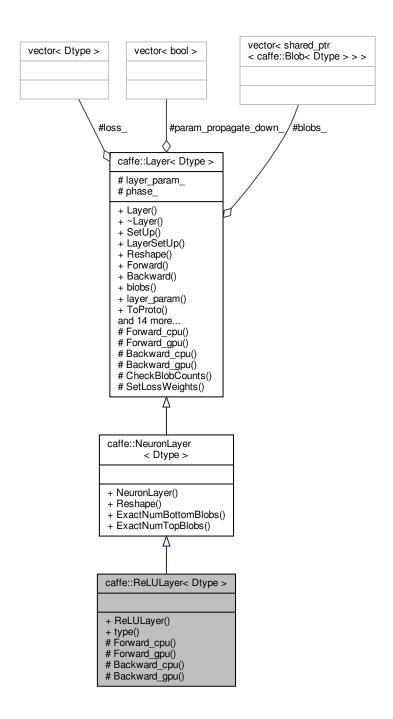
Rectified Linear Unit non-linearity  $y=\max(0,x)$ . The simple max is fast to compute, and the function does not saturate.

```
#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()

## **Parameters**

param

provides ReLUParameter relu\_param, with ReLULayer options:

• negative\_slope (**optional**, default 0). the value  $\nu$  by which negative values are multiplied.

## 5.77.3 Member Function Documentation

## 5.77.3.1 Backward\_cpu()

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 $\frac{\partial E}{\partial x} = \begin{cases} 0 & \text{if } x \leq 0 \\ \frac{\partial E}{\partial y} & \text{if } x > 0 \end{cases}$ if propagate_down[0], by default. If a non-zero negative_slope $\nu$ is provided, the computed gradients are $\frac{\partial E}{\partial x} = \begin{cases} \nu \frac{\partial E}{\partial y} & \text{if } x \leq 0 \\ \frac{\partial E}{\partial y} & \text{if } x > 0 \end{cases}.$

Implements caffe::Layer< Dtype >.

## 5.77.3.2 Forward\_cpu()

## **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 $\nu$ 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:

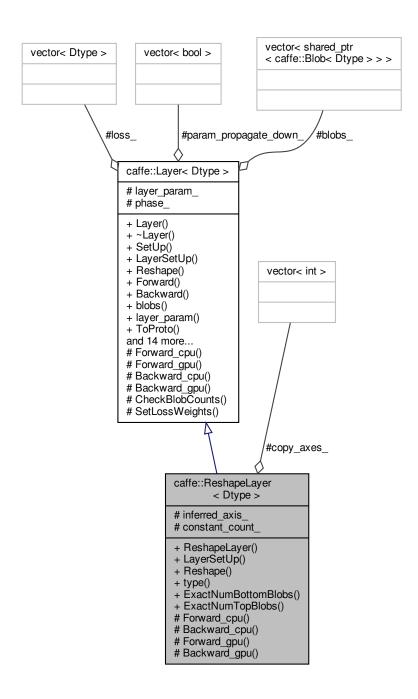
- include/caffe/layers/relu\_layer.hpp
- src/caffe/layers/relu\_layer.cpp

## 5.78 caffe::ReshapeLayer < Dtype > Class Template Reference

Inheritance diagram for caffe::ReshapeLayer< Dtype >:

# caffe::Layer< Dtype > # layer\_param\_ # phase\_ # blobs\_ # param\_propagate\_down\_ # loss\_ + Layer() + ~Layer() + SetÚp() + LayerSetUp() + Reshape() + Forward() + Backward() + blobs() + layer\_param() + ToProto() and 14 more... # Forward\_cpu() # Forward\_gpu() # Backward\_cpu() # Backward\_gpu() # CheckBlobCounts() # SetLossWeights() caffe::ReshapeLayer < Dtype > # copy\_axes\_ # inferred axis # constant\_count\_ + ReshapeLayer() + LayerSetUp() + Reshape() + type() + ExactNumBottomBlobs() + ExactNumTopBlobs() # Forward\_cpu() # Backward\_cpu() # Forward\_gpu() # Backward\_gpu()

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()

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
ſ	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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

#### 5.78.1.4 Reshape()

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

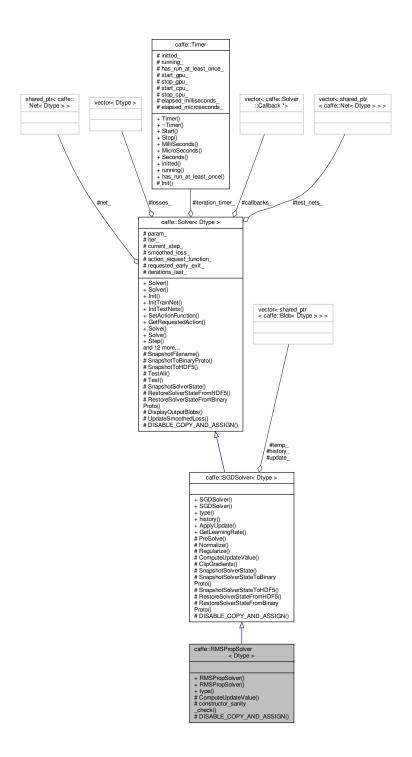
reshaping any internal buffers and making any other necessary adjustments so that the layer can accommodate th bottom blobs.
Implements caffe::Layer< Dtype >.
The documentation for this class was generated from the following files:
• include/caffe/layers/reshape_layer.hpp
• Include/carie/layers/reshape_layer.hpp
• 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:

+ RNG() + RNG() + RNG() + RNG() + operator=() + generator()

# 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:

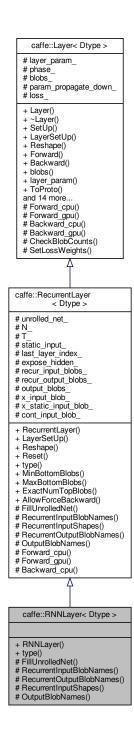
- include/caffe/common.hpp
- src/caffe/common.cpp

# 5.81 caffe::RNNLayer < Dtype > Class Template Reference

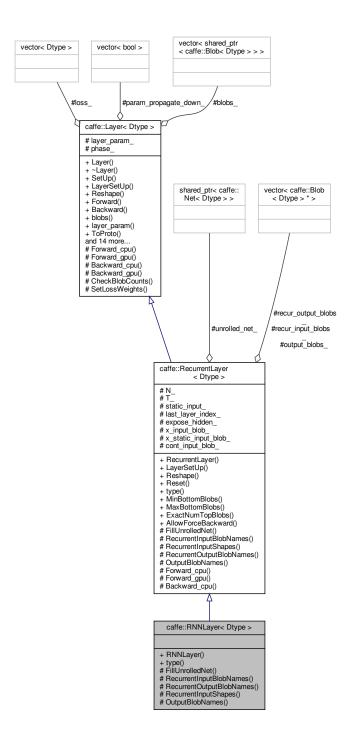
Processes time-varying inputs using a simple recurrent neural network (RNN). Implemented as a network unrolling the RNN computation in time.

```
#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 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**

### 5.81.1 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

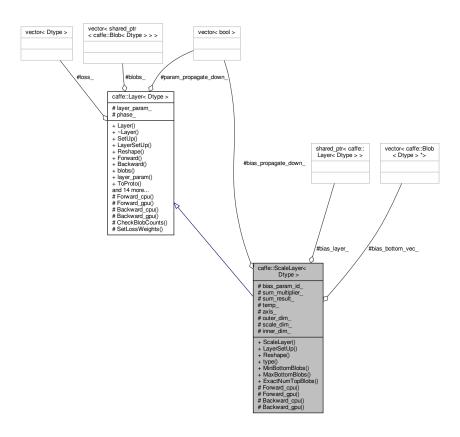
Computes the elementwise product of two input Blobs, with the shape of the latter Blob "broadcast" to match the shape of the former. Equivalent to tiling the latter Blob, then computing the elementwise product. Note: for efficiency and convenience, this layer can additionally perform a "broadcast" sum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

```
#include <scale_layer.hpp>
```

Inheritance diagram for caffe::ScaleLayer< Dtype >:

```
caffe::Layer< Dtype >
# layer_param_
# phase_
# blobs
# param_propagate_down_
# loss
+ Layer()
+ ~Layer()
+ SetUp()
+ LayerSetUp()
+ Reshape()
+ Forward()
+ Folward()
+ Backward()
+ blobs()
+ layer_param()
+ ToProto()
and 14 more...
# Forward_cpu()
# Forward_gpu()
# Backward_cpu()
# Backward_gpu()
# CheckBlobCounts()
# Set less Weights()
# SetLossWeights()
  caffe::ScaleLayer<
                 Dtype >
 # bias_layer_
# bias_bottom_vec_
# bias_propagate_down_
  # bias_param_id_
  # sum_multiplier_
  # sum_result_
# temp_
 # axis_
# outer_dim_
# scale_dim_
# inner_dim_
  + ScaleLayer()
+ LayerSetUp()
+ Reshape()
  + type()
+ MinBottomBlobs()
+ MaxBottomBlobs()
  + MaxBottomBlobs()
+ ExactNumTopBlobs()
# Forward_cpu()
# Forward_gpu()
# Backward_cpu()
# Backward_gpu()
```

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()

In the below shape specifications, i denotes the value of the axis field given by this->layer\_param\_.  $\leftarrow$  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()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, 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()

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:

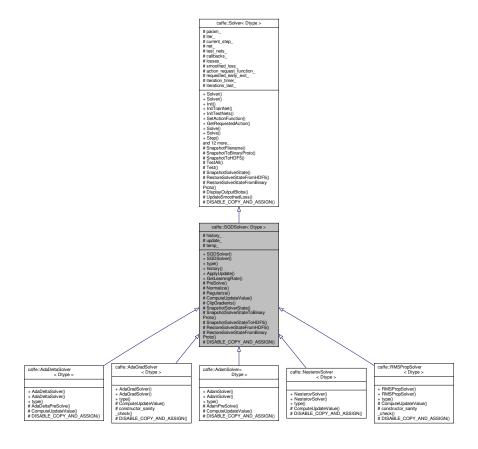
- include/caffe/layers/scale\_layer.hpp
- src/caffe/layers/scale\_layer.cpp

# 5.83 caffe::SGDSolver< Dtype > Class Template Reference

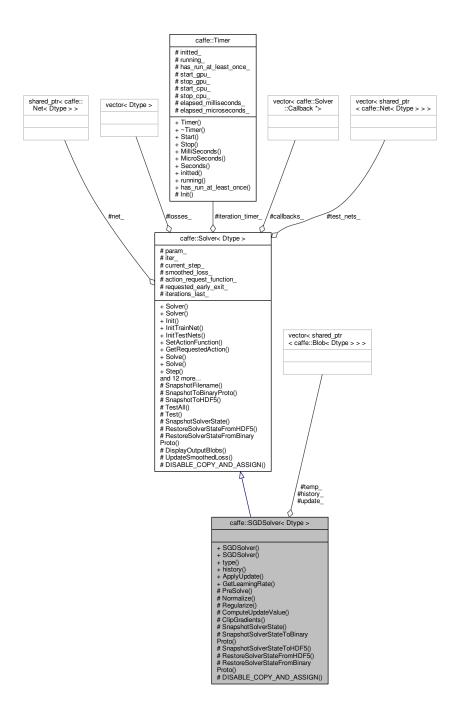
Optimizes the parameters of a Net using stochastic gradient descent (SGD) with momentum.

#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:

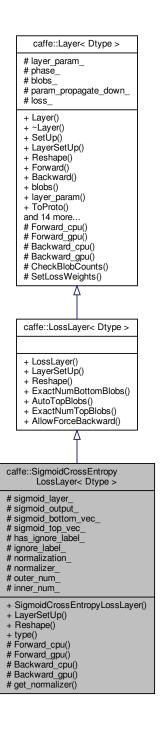
- include/caffe/sgd\_solvers.hpp
- src/caffe/solvers/sgd\_solver.cpp

# 5.84 caffe::SigmoidCrossEntropyLossLayer < Dtype > Class Template Reference

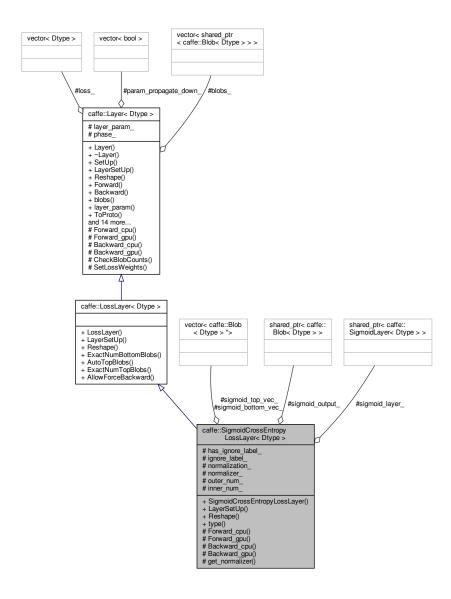
Computes the cross-entropy (logistic) loss  $E = \frac{-1}{n} \sum_{n=1}^{N} \left[ p_n \log \hat{p}_n + (1-p_n) \log (1-\hat{p}_n) \right]$ , often used for predicting targets interpreted as probabilities.

#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  $E = \frac{-1}{n} \sum_{n=1}^{N} [p_n \log \hat{p}_n + (1-p_n) \log (1-\hat{p}_n)]$ , 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  $E = \frac{-1}{n} \sum_{n=1}^{N} [p_n \log \hat{p}_n + (1-p_n) \log (1-\hat{p}_n)]$ , 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:
	$E = \frac{-1}{n} \sum_{n=1}^{N} \left[ p_n \log \hat{p}_n + (1 - p_n) \log(1 - \hat{p}_n) \right]$

# 5.84.2 Member Function Documentation

# 5.84.2.1 Backward\_cpu()

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* $\lambda$ , as $\lambda$ is the coefficient of this layer's output $\ell_i$ 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 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 $\frac{\partial E}{\partial x} = \frac{1}{n} \sum_{n=1}^{N} (\hat{p}_n - p_n)$
	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()

Computes the cross-entropy (logistic) loss  $E = \frac{-1}{n} \sum_{n=1}^{N} \left[ p_n \log \hat{p}_n + (1-p_n) \log (1-\hat{p}_n) \right]$ , 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:
	$E = \frac{-1}{n} \sum_{n=1}^{N} \left[ p_n \log \hat{p}_n + (1 - p_n) \log(1 - \hat{p}_n) \right]$

Implements caffe::Layer< Dtype >.

#### 5.84.2.3 get\_normalizer()

Read the normalization mode parameter and compute the normalizer based on the blob size. If normalization\_\circ} 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()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::LossLayer< Dtype >.

#### 5.84.2.5 Reshape()

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

#### **Parameters**

botte	om	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:

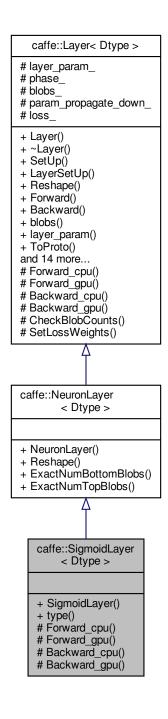
- include/caffe/layers/sigmoid\_cross\_entropy\_loss\_layer.hpp
- src/caffe/layers/sigmoid\_cross\_entropy\_loss\_layer.cpp

# 5.85 caffe::SigmoidLayer < Dtype > Class Template Reference

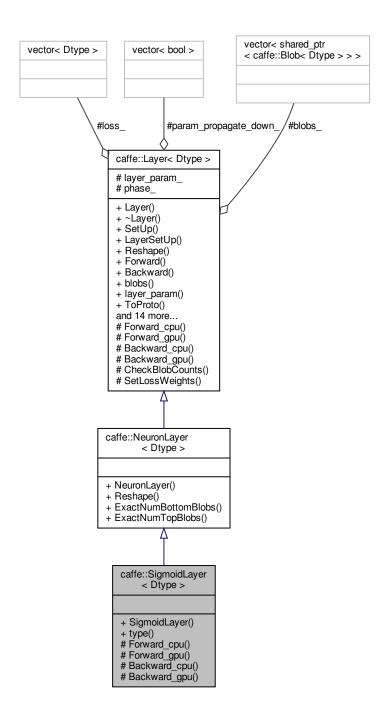
Sigmoid function non-linearity  $y = (1 + \exp(-x))^{-1}$ , a classic choice in neural networks.

```
#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()

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]
	Generated by Doxygen

Implements caffe::Layer< Dtype >.

#### 5.85.2.2 Forward\_cpu()

#### **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:

+ SignalHandler()
+ ~SignalHandler()
+ GetActionFunction()

### **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:

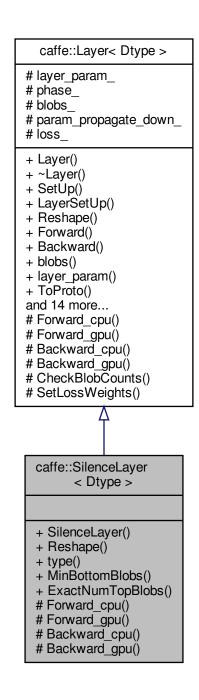
- · include/caffe/util/signal\_handler.h
- src/caffe/util/signal\_handler.cpp

# 5.87 caffe::SilenceLayer < Dtype > Class Template Reference

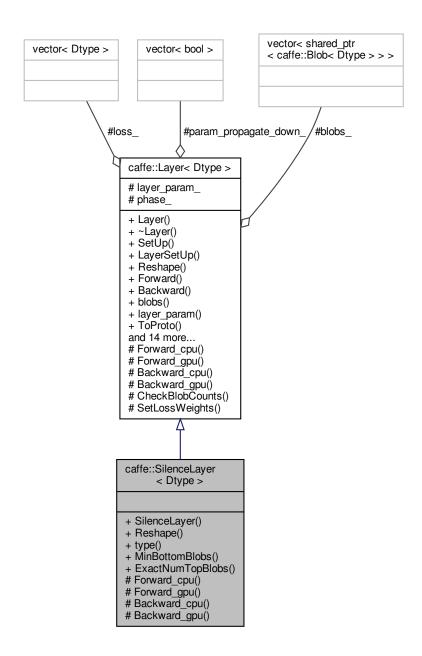
Ignores bottom blobs while producing no top blobs. (This is useful to suppress outputs during testing.)

```
#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

```
\label{lem:continuous} \mbox{template}{<} \mbox{typename Dtype}{>} \\ \mbox{class caffe::SilenceLayer}{<} \mbox{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()

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

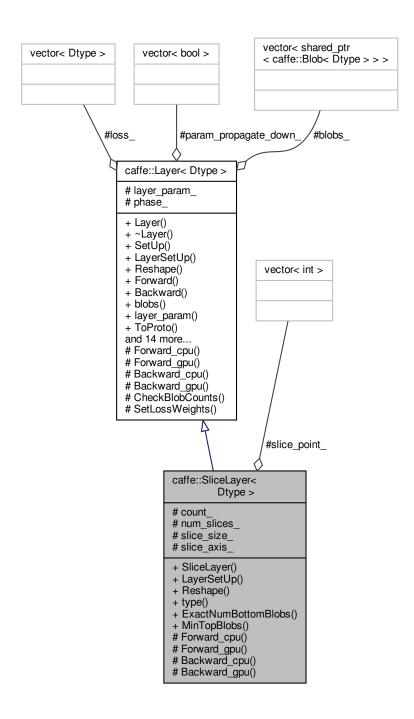
Takes a Blob and slices it along either the num or channel dimension, outputting multiple sliced Blob results.

```
#include <slice_layer.hpp>
```

Inheritance diagram for caffe::SliceLayer< Dtype >:

# caffe::Layer< Dtype > # layer\_param\_ # phase\_ # blobs # param\_propagate\_down\_ # loss + Layer() + ~Layer() + SetUp() + LayerSetUp() + Reshape() + Forward() + Backward() + blobs() + layer\_param() + ToProto() and 14 more... # Forward\_cpu() # Forward\_gpu() # Backward\_cpu() # Backward\_gpu() # CheckBlobCounts() # SetLossWeights() caffe::SliceLayer< Dtype > # count\_ # num\_slices\_ # slice\_size\_ # slice\_axis\_ # slice\_point\_ + SliceLayer() + LayerSetUp() + Reshape() + Heshape() + type() + ExactNumBottomBlobs() + MinTopBlobs() # Forward\_cpu() # Forward\_gpu() # Backward\_cpu() # Backward\_gpu()

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()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, 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()

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

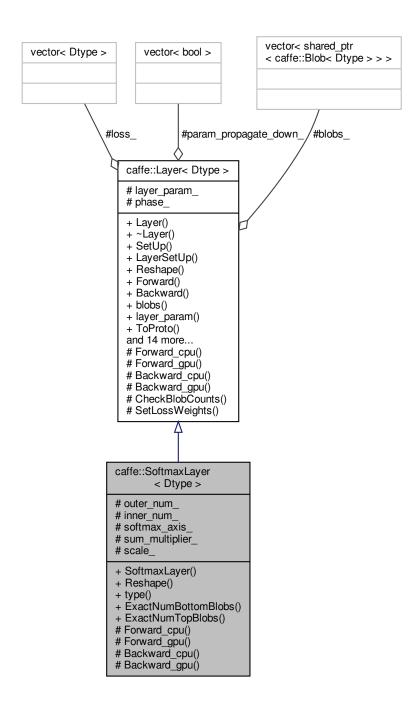
Computes the softmax function.

#include <softmax\_layer.hpp>

Inheritance diagram for caffe::SoftmaxLayer< Dtype >:

# caffe::Layer< Dtype > # layer\_param\_ # phase\_ # blobs\_ # param\_propagate\_down\_ # İoss\_ + Layer() + ~Layer() + SetUp() + LayerSetUp() + Reshape() + Forward() + Backward() + blobs() + layer\_param() + ToProto() and 14 more... # Forward\_cpu() # Forward\_gpu() # Backward\_cpu() # Backward\_gpu() # CheckBlobCounts() # SetLossWeights() caffe::SoftmaxLayer < Dtype > # outer num # inner\_num\_ # softmax\_axis\_ # sum\_multiplier\_ # scale\_ + SoftmaxLayer() + Reshape() + type() + ÉxactNumBottomBlobs() + ExactNumTopBlobs() # Forward\_cpu() # Forward\_gpu() # Backward\_cpu() # Backward\_gpu()

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()

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:

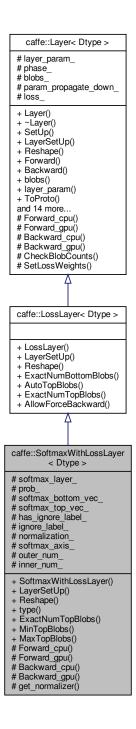
- include/caffe/layers/softmax\_layer.hpp
- src/caffe/layers/softmax\_layer.cpp

# 5.90 caffe::SoftmaxWithLossLayer < Dtype > Class Template Reference

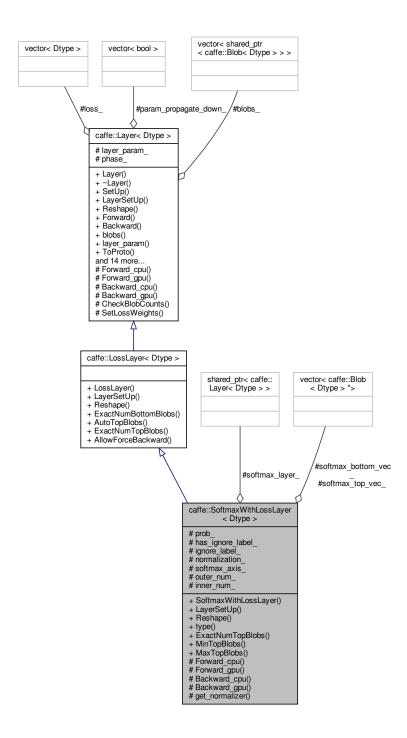
Computes the multinomial logistic loss for a one-of-many classification task, passing real-valued predictions through a softmax to get a probability distribution over classes.

```
#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

 $\label{lem:continuous} \mbox{template} \! < \! \mbox{typename Dtype} \! > \\ \mbox{class caffe} \! :: \! \mbox{SoftmaxWithLossLayer} \! < \! \mbox{Dtype} > \\ \mbox{}$ 

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: $E = \frac{-1}{N} \sum_{n=1}^{N} \log(\hat{p}_{n,l_n})$ , for softmax output class probabilites $\hat{p}$

#### 5.90.2 Constructor & Destructor Documentation

# 5.90.2.1 SoftmaxWithLossLayer()

#### **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()

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 $ \text{1. } (1\times 1\times 1\times 1) \text{ This Blob's diff will simply contain the loss\_weight* } \lambda \text{, as } \lambda \text{ is the } $
	coefficient of this layer's output $\ell_i$ 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()

Read the normalization mode parameter and compute the normalizer based on the blob size. If normalization\_\circ} 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()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, 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()

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:

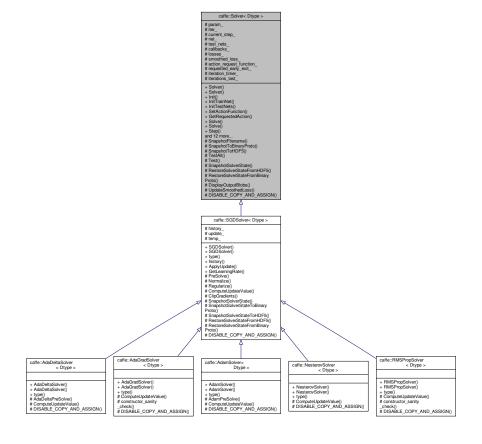
- include/caffe/layers/softmax\_loss\_layer.hpp
- src/caffe/layers/softmax\_loss\_layer.cpp

# 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

```
\label{template} \begin{tabular}{ll} template < typename \ Dtype > \\ class \ caffe:: Solver < Dtype > \\ \end{tabular}
```

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 >:

caffe::SolverRegistry < Dtype > + Registry() + AddCreator() + CreateSolver() + SolverTypeList()

# **Public Types**

- typedef Solver<br/>< 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

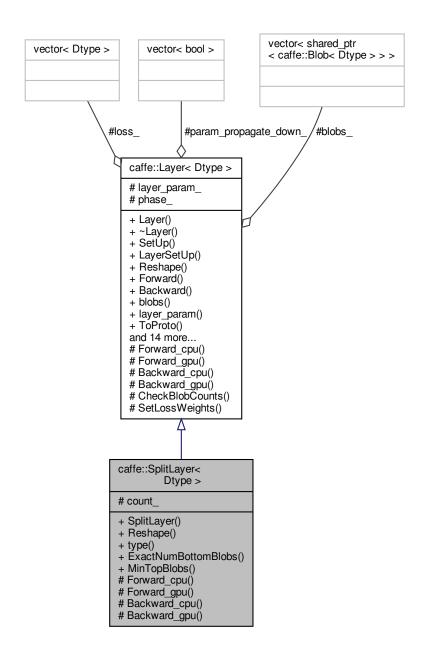
Creates a "split" path in the network by copying the bottom Blob into multiple top Blobs to be used by multiple consuming layers.

#include <split\_layer.hpp>

Inheritance diagram for caffe::SplitLayer< Dtype >:

# caffe::Layer< Dtype > # layer\_param\_ # phase # blobs\_ # param\_propagate\_down\_ # loss + Layer() + ~Layer() + SetUp() + LayerSetUp() + Reshape() + Forward() + Backward() + blobs() + layer\_param() + ToProto() and 14 more... # Forward\_cpu() # Forward\_gpu() # Backward\_cpu() # Backward\_gpu() # CheckBlobCounts() # SetLossWeights() caffe::SplitLayer< Dtype > # count\_ + SplitLayer() + Reshape() + type() + ExactNumBottomBlobs() + MinTopBlobs() # Forward\_cpu() # Forward\_gpu() # Backward\_cpu() # Backward\_gpu()

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()

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

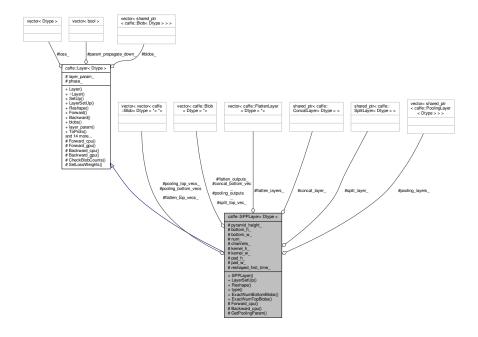
Does spatial pyramid pooling on the input image by taking the max, average, etc. within regions so that the result vector of different sized images are of the same size.

```
#include <spp_layer.hpp>
```

Inheritance diagram for caffe::SPPLayer< Dtype >:

```
caffe::Layer< Dtype >
# layer_param_
# phase_
# blobs
# param_propagate_down_
# loss_
+ Layer()
+ ~Layer()
+ SetUp()
+ LayerSetUp()
+ Reshape()
+ Forward()
+ Backward()
+ blobs()
+ layer_param()
+ ToProto()
and 14 more...
# Forward_cpu()
# Forward_gpu()
# Backward_cpu()
# Backward_gpu()
# CheckBlobCounts()
# SetLossWeights()
caffe::SPPLayer< Dtype >
# pyramid_height_
# bottom_h_
# bottom_w_
# num_
# channels_
# kernel_h_
# kernel_w_
# pad h
# pad w
# reshaped_first_time_
and 11 more...
+ SPPLayer()
+ LayerSetUp()
+ Reshape()
+ Hesnape()
+ type()
+ ExactNumBottomBlobs()
+ ExactNumTopBlobs()
# Forward_cpu()
# Backward_cpu()
# GetPoolingParam()
```

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</li>
 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()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

# 5.95.2.4 Reshape()

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:

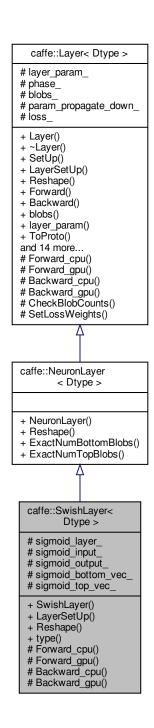
- include/caffe/layers/spp\_layer.hpp
- src/caffe/layers/spp\_layer.cpp

# 5.96 caffe::SwishLayer < Dtype > Class Template Reference

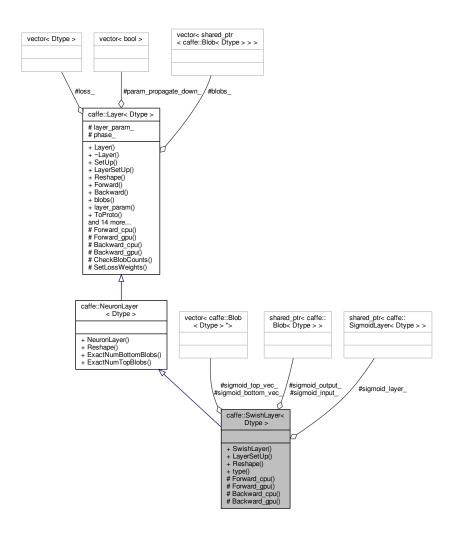
Swish non-linearity  $y = x\sigma(\beta x)$ . A novel activation function that tends to work better than ReLU [1].

```
#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.

sigmoid\_output\_ stores the output of the SigmoidLayer.

vector< Blob</li>
 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()

# **Parameters**

# param provides SwishParameter swish\_param, with SwishLayer options:

• beta (**optional**, default 1). the value  $\beta$  in the  $y = x\sigma(\beta x)$ .

# 5.96.3 Member Function Documentation

# 5.96.3.1 Backward\_cpu()

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 $\frac{\partial E}{\partial x} = \frac{\partial E}{\partial y}(\beta y + \sigma(\beta x)(1-\beta y)) \text{ if propagate\_down[0]}$

Implements caffe::Layer < Dtype >.

# 5.96.3.2 Forward\_cpu()

# **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()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, which will be called before the forward pass to adjust the top blob sizes.

Reimplemented from caffe::Layer< Dtype >.

# 5.96.3.4 Reshape()

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:

caffe::BlockingQueue < T >::sync + mutex\_ + condition\_

# **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:

+ SyncedMemory()
+ SyncedMemory()
+ SyncedMemory()
+ ~SyncedMemory()
+ cpu\_data()
+ set\_cpu\_data()
+ gpu\_data()
+ mutable\_cpu\_data()
+ mutable\_gpu\_data()
+ head()
+ size()
+ async\_gpu\_push()

# **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:

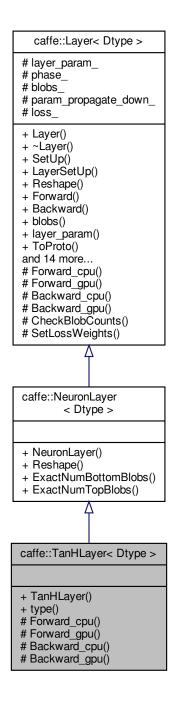
- include/caffe/syncedmem.hpp
- src/caffe/syncedmem.cpp

# 5.99 caffe::TanHLayer< Dtype > Class Template Reference

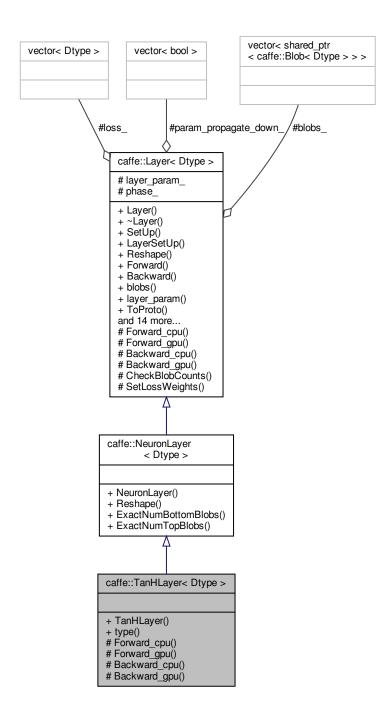
TanH hyperbolic tangent non-linearity  $y=rac{\exp(2x)-1}{\exp(2x)+1}$ , popular in auto-encoders.

#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  $y=rac{\exp(2x)-1}{\exp(2x)+1}$ , 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()

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 $\frac{\partial E}{\partial x} = \frac{\partial E}{\partial y} \left(1 - \left[\frac{\exp(2x) - 1}{\exp(2x) + 1}\right]^2\right) = \frac{\partial E}{\partial y} (1 - y^2) \text{ if propagate\_down[0]}_{\text{Generated by Doxygen}}$

Implements caffe::Layer < Dtype >.

#### 5.99.2.2 Forward\_cpu()

#### **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:

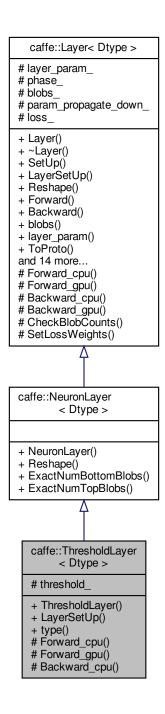
- include/caffe/layers/tanh\_layer.hpp
- src/caffe/layers/tanh\_layer.cpp

# 5.100 caffe::ThresholdLayer < Dtype > Class Template Reference

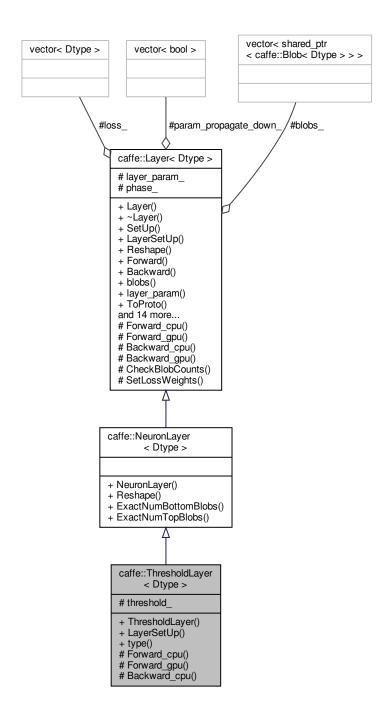
Tests whether the input exceeds a threshold: outputs 1 for inputs above threshold; 0 otherwise.

```
#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()

# **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 $y = \left\{ \begin{array}{ll} 0 & \text{if } x \leq t \\ 1 & \text{if } x > t \end{array} \right.$	

Implements caffe::Layer< Dtype >.

# 5.100.3.2 LayerSetUp()

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 <code>layer\_param\_</code>. Setting up the shapes of top blobs and internal buffers should be done in <code>Reshape</code>, 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

# ${\bf 5.101} \quad {\bf caffe::TileLayer} < {\bf Dtype} > {\bf Class\ Template\ Reference}$

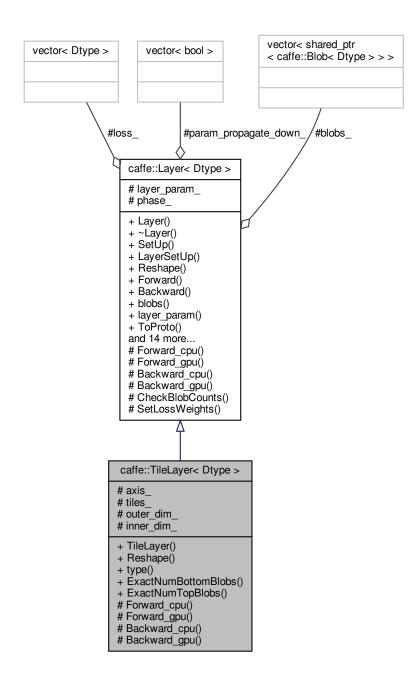
Copy a Blob along specified dimensions.

```
#include <tile_layer.hpp>
```

Inheritance diagram for caffe::TileLayer< Dtype >:

```
caffe::Layer< Dtype >
# layer_param_
# phase
# blobs_
# param_propagate_down_
# loss_
+ Layer()
+ ~Layer()
+ SetUp()
+ LayerSetUp()
+ Reshape()
+ Forward()
+ Backward()
+ blobs()
+ layer_param()
+ ToProto()
and 14 more...
# Forward_cpu()
# Forward_gpu()
# Backward_cpu()
# Backward_gpu()
# CheckBlobCounts()
# SetLossWeights()
caffe::TileLayer< Dtype >
# axis_
# tiles_
# outer_dim_
# inner_dim_
+ TileLayer()
+ Reshape()
+ type()
+ ExactNumBottomBlobs()
+ ExactNumTopBlobs()
# Forward_cpu()
# Forward_gpu()
# Backward_cpu()
# Backward_gpu()
```

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()

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.

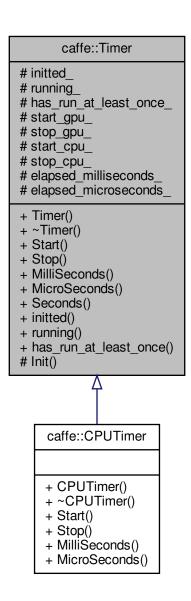
Implements caffe::Layer< Dtype >.

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:

```
caffe::Timer
# initted_
# running_
# has_run_at_least_once_
# start_gpu_
# stop_gpu_
# start_cpu_
# stop_cpu_
# elapsed_milliseconds_
# elapsed_microseconds_
+ Timer()
+ ~Timer()
+ Start()
+ Stop()
+ MilliSeconds()
+ MicroSeconds()
+ Seconds()
+ initted()
+ running()
+ has_run_at_least_once()
# Init()
```

# **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:

# + Transaction() + ~Transaction() + ~Transaction() + Put() + Commit() + DISABLE\_COPY\_AND\_ASSIGN()

# **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:

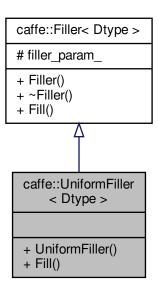
include/caffe/util/db.hpp

# 5.104 caffe::UniformFiller < Dtype > Class Template Reference

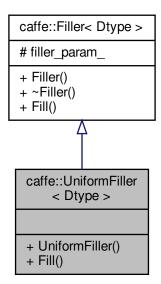
Fills a Blob with uniformly distributed values  $x \sim U(a, b)$ .

#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:

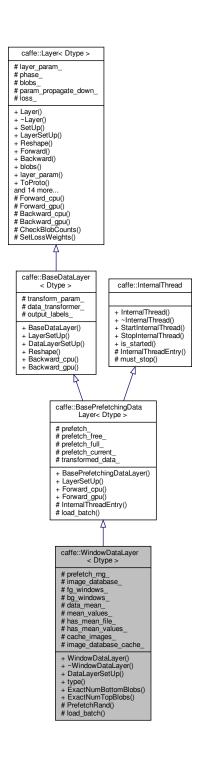
• include/caffe/filler.hpp

# 5.105 caffe::WindowDataLayer < Dtype > Class Template Reference

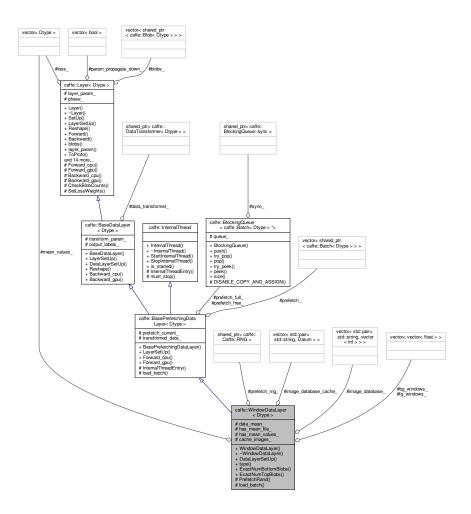
Provides data to the Net from windows of images files, specified by a window data file. This layer is *DEPRECATED* and only kept for archival purposes for use by the original R-CNN.

#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:

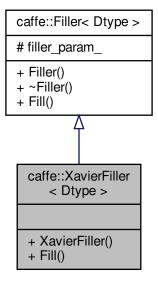
• include/caffe/layers/window\_data\_layer.hpp

# 5.106 caffe::XavierFiller < Dtype > Class Template Reference

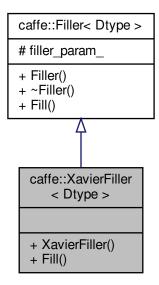
Fills a Blob with values  $x \sim U(-a, +a)$  where a is set inversely proportional to number of incoming nodes, outgoing nodes, or their average.

```
#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

AccuracyLayer	caffe, 13
caffe::AccuracyLayer, 39	GetFiller, 28
AllowForceBackward	caffe::AbsValLayer
caffe::ContrastiveLossLayer, 117	Backward_cpu, 34
caffe::EuclideanLossLayer, 167	ExactNumBottomBlobs, 35
caffe::LSTMUnitLayer, 264	ExactNumTopBlobs, 35
caffe::Layer, 236	Forward_cpu, 35
caffe::LossLayer, 251	LayerSetUp, 36
caffe::RecurrentLayer, 322	caffe::AbsValLayer< Dtype >, 31
ArgMaxLayer	caffe::AccuracyLayer
caffe::ArgMaxLayer, 54	AccuracyLayer, 39
AutoTopBlobs	ExactNumBottomBlobs, 40
caffe::Layer, 236	Forward_cpu, 40
	LayerSetUp, 41
Backward	MaxTopBlobs, 41
caffe::Layer, 236	MinTopBlobs, 41
caffe::Net, 288	Reshape, 41
Backward_cpu	caffe::AccuracyLayer< Dtype >, 36
caffe::AbsValLayer, 34	caffe::AdaDeltaSolver< Dtype >, 43
caffe::BNLLLayer, 98	caffe::AdaGradSolver< Dtype >, 46
caffe::BatchReindexLayer, 78	caffe::AdamSolver< Dtype >, 48
caffe::ClipLayer, 105	caffe::ArgMaxLayer
caffe::ConcatLayer, 109	ArgMaxLayer, 54
caffe::ContrastiveLossLayer, 117	ExactNumBottomBlobs, 55
caffe::ELULayer, 157	ExactNumTopBlobs, 55
caffe::EuclideanLossLayer, 167	Forward_cpu, 55
caffe::ExpLayer, 173	LayerSetUp, 56
caffe::FilterLayer, 178	Reshape, 56
caffe::FlattenLayer, 184	caffe::ArgMaxLayer< Dtype >, 51
caffe::HingeLossLayer, 202	caffe::BNLLLayer
caffe::InfogainLossLayer, 216	Backward_cpu, 98
caffe::LSTMUnitLayer, 264	Forward_cpu, 99
caffe::LogLayer, 248	caffe::BNLLLayer< Dtype >, 95
caffe::MultinomialLogisticLossLayer, 275	caffe::BaseConvolutionLayer
caffe::PReLULayer, 312	EqualNumBottomTopBlobs, 61
caffe::PowerLayer, 307	LayerSetUp, 61
caffe::ReLULayer, 333	MinBottomBlobs, 61
caffe::SigmoidCrossEntropyLossLayer, 357	MinTopBlobs, 62
caffe::SigmoidLayer, 362	Reshape, 62
caffe::SoftmaxWithLossLayer, 382	caffe::BaseConvolutionLayer< Dtype >, 57
caffe::SwishLayer, 402	caffe::BaseDataLayer
caffe::TanHLayer, 408	LayerSetUp, 65
blob_loss_weights_	Reshape, 65
caffe::Net, 289	caffe::BaseDataLayer< Dtype >, 63
blobs_	caffe::BasePrefetchingDataLayer
caffe::Layer, 242	LayerSetUp, 69
boost, 13	caffe::BasePrefetchingDataLayer< Dtype >, 6
bottom_vecs_	caffe::Batch < Dtype >, 69
caffe::Net, 289	caffe::BatchNormLayer

ExactNumBottomBlobs, 74	caffe::ConvolutionLayer< Dtype >, 119
ExactNumTopBlobs, 74	caffe::CropLayer
LayerSetUp, 74	ExactNumBottomBlobs, 128
Reshape, 75	ExactNumTopBlobs, 128
caffe::BatchNormLayer< Dtype >, 70	LayerSetUp, 129
caffe::BatchReindexLayer	Reshape, 129
Backward_cpu, 78	caffe::CropLayer< Dtype >, 126
ExactNumBottomBlobs, 79	caffe::DataLayer
ExactNumTopBlobs, 79	ExactNumBottomBlobs, 133
Forward_cpu, 79	MaxTopBlobs, 133
Reshape, 80	MinTopBlobs, 133
caffe::BatchReindexLayer< Dtype >, 75	caffe::DataLayer< Dtype >, 131
caffe::BiasLayer	caffe::DataTransformer
ExactNumTopBlobs, 83	InferBlobShape, 135
LayerSetUp, 83	Rand, 136
MaxBottomBlobs, 85	Transform, 136, 137
MinBottomBlobs, 85	caffe::DataTransformer< Dtype >, 134
Reshape, 85	caffe::DeconvolutionLayer $<$ Dtype $>$ , 138
caffe::BiasLayer< Dtype >, 80	caffe::DropoutLayer
caffe::BilinearFiller< Dtype >, 86	DropoutLayer, 144
caffe::Blob	Forward_cpu, 145
CanonicalAxisIndex, 91	LayerSetUp, 145
CopyFrom, 91	Reshape, 146
count, 92	caffe::DropoutLayer< Dtype >, 141
Reshape, 92	caffe::DummyDataLayer
shape, 93	ExactNumBottomBlobs, 149
ShareData, 93	LayerSetUp, 149
ShareDiff, 93	MinTopBlobs, 150
caffe::Blob < Dtype >, 88	Reshape, 150
caffe::BlockingQueue< T >, 94	caffe::DummyDataLayer< Dtype >, 146
caffe::BlockingQueue< T >::sync, 404	caffe::ELULayer
caffe::CPUTimer, 124	Backward_cpu, 157
caffe::Caffe, 100	ELULayer, 157
caffe::Caffe::RNG::Generator, 188	Forward_cpu, 158
caffe::Caffe::RNG, 342	caffe::ELULayer< Dtype >, 155
caffe::ClipLayer	caffe::EltwiseLayer
Backward_cpu, 105	ExactNumTopBlobs, 153
ClipLayer, 105	LayerSetUp, 153
Forward_cpu, 106	MinBottomBlobs, 154
caffe::ClipLayer< Dtype >, 103	Reshape, 154
caffe::ConcatLayer	caffe::EltwiseLayer< Dtype >, 151
Backward_cpu, 109	caffe::EmbedLayer
ExactNumTopBlobs, 110	ExactNumBottomBlobs, 161
Forward_cpu, 110	ExactNumTopBlobs, 162
LayerSetUp, 111	LayerSetUp, 162
MinBottomBlobs, 111	Reshape, 162
Reshape, 111	caffe::EmbedLayer< Dtype >, 159
caffe::ConcatLayer< Dtype >, 107	caffe::EuclideanLossLayer
caffe::ConstantFiller< Dtype >, 112	AllowForceBackward, 167
caffe::ContrastiveLossLayer	Backward_cpu, 167
AllowForceBackward, 117	Forward_cpu, 168
Backward_cpu, 117	Reshape, 168
ExactNumBottomBlobs, 117	caffe::EuclideanLossLayer< Dtype >, 163
Forward_cpu, 118	caffe::ExpLayer
LayerSetUp, 118	Backward_cpu, 173
caffe::ContrastiveLossLayer< Dtype >, 114	ExpLayer, 172
caffe::ConvolutionLayer	Forward_cpu, 173
ConvolutionLayer, 122	LayerSetUp, 173

caffe::ExpLayer< Dtype >, 169	ExactNumBottomBlobs, 224
caffe::Filler< Dtype >, 174	ExactNumTopBlobs, 225
caffe::FilterLayer	LayerSetUp, 225
Backward_cpu, 178	Reshape, 225
Forward_cpu, 179	caffe::InnerProductLayer< Dtype >, 22
LayerSetUp, 179	caffe::InputLayer
MinBottomBlobs, 180	ExactNumBottomBlobs, 229
MinTopBlobs, 180	LayerSetUp, 229
Reshape, 180	MinTopBlobs, 230
caffe::FilterLayer< Dtype >, 175	Reshape, 230
caffe::FlattenLayer	caffe::InputLayer< Dtype >, 226
Backward_cpu, 184	caffe::InternalThread, 231
ExactNumBottomBlobs, 184	StartInternalThread, 232
ExactNumTopBlobs, 185	StopInternalThread, 232
Forward_cpu, 185	caffe::LRNLayer
Reshape, 185	ExactNumBottomBlobs, 257
caffe::FlattenLayer< Dtype >, 181	ExactNumTopBlobs, 257
caffe::GaussianFiller< Dtype >, 186	LayerSetUp, 257
caffe::HDF5DataLayer	Reshape, 258
ExactNumBottomBlobs, 191	caffe::LRNLayer< Dtype >, 253
LayerSetUp, 192	caffe::LSTMLayer< Dtype >, 258
MinTopBlobs, 192	caffe::LSTMUnitLayer
Reshape, 192	AllowForceBackward, 264
caffe::HDF5DataLayer< Dtype >, 188	Backward_cpu, 264
caffe::HDF5OutputLayer	ExactNumBottomBlobs, 265
ExactNumBottomBlobs, 197	ExactNumTopBlobs, 265
ExactNumTopBlobs, 198	Forward_cpu, 265
LayerSetUp, 198	Reshape, 266
Reshape, 198	caffe::LSTMUnitLayer< Dtype >, 262
caffe::HDF5OutputLayer< Dtype >, 194	caffe::Layer
caffe::HingeLossLayer	AllowForceBackward, 236
Backward_cpu, 202	AutoTopBlobs, 236
Forward_cpu, 203	Backward, 236
caffe::HingeLossLayer< Dtype >, 199	blobs_, 242
caffe::Im2colLayer	CheckBlobCounts, 237
ExactNumBottomBlobs, 207	EqualNumBottomTopBlobs, 237
ExactNumTopBlobs, 208	ExactNumBottomBlobs, 237
LayerSetUp, 208	ExactNumTopBlobs, 237
Reshape, 208	Forward, 238
caffe::Im2colLayer< Dtype >, 204	Layer, 235
caffe::ImageDataLayer	layer_param_, 242
ExactNumBottomBlobs, 212	LayerSetUp, 238
ExactNumTopBlobs, 212	loss_, 242
caffe::ImageDataLayer< Dtype >, 209	MaxBottomBlobs, 239
caffe::InfogainLossLayer	MaxTopBlobs, 239
Backward_cpu, 216	MinBottomBlobs, 239
ExactNumBottomBlobs, 217	MinTopBlobs, 240
ExactNumTopBlobs, 217	param_propagate_down, 240
Forward_cpu, 217	param_propagate_down_, 242
get_normalizer, 219	phase_, 242
LayerSetUp, 219	Reshape, 240
MaxBottomBlobs, 220	SetLossWeights, 241
MaxTopBlobs, 220	SetUp, 241
MinBottomBlobs, 220	caffe::Layer< Dtype >, 233
MinTopBlobs, 220	caffe::LayerRegisterer< Dtype >, 243
Reshape, 221	caffe::LayerRegistry< Dtype >, 243
caffe::InfogainLossLayer< Dtype >, 213	caffe::LogLayer
caffe::InnerProductLayer	Backward_cpu, 248

Forward_cpu, 248	MaxTopBlobs, 300
LayerSetUp, 248	MinTopBlobs, 300
LogLayer, 247	Reshape, 301
caffe::LogLayer< Dtype >, 244	caffe::PoolingLayer< Dtype >, 296
caffe::LossLayer	caffe::PositiveUnitballFiller< Dtype >, 30
AllowForceBackward, 251	caffe::PowerLayer
ExactNumBottomBlobs, 251	Backward_cpu, 307
ExactNumTopBlobs, 251	Forward_cpu, 307
LayerSetUp, 252	LayerSetUp, 308
Reshape, 252	PowerLayer, 306
caffe::LossLayer< Dtype >, 249	caffe::PowerLayer< Dtype >, 303
caffe::MSRAFiller< Dtype >, 270	caffe::PythonLayer
caffe::MVNLayer	LayerSetUp, 317
ExactNumBottomBlobs, 280	Reshape, 317
ExactNumTopBlobs, 281	caffe::PythonLayer< Dtype >, 315
Reshape, 281	caffe::RMSPropSolver< Dtype >, 340
caffe::MVNLayer< Dtype >, 277	caffe::RNNLayer< Dtype >, 343
caffe::MemoryDataLayer	caffe::ReLULayer
ExactNumBottomBlobs, 269	Backward_cpu, 333
	Forward cpu, 334
ExactNumTopBlobs, 270	ReLULayer, 333
caffe::MemoryDataLayer< Dtype >, 267	• •
caffe::MultinomialLogisticLossLayer	caffe::ReLULayer< Dtype >, 330
Backward_cpu, 275	caffe::RecurrentLayer
Forward_cpu, 276	AllowForceBackward, 322
Reshape, 277	ExactNumTopBlobs, 322
caffe::MultinomialLogisticLossLayer< Dtype >, 272	Forward_cpu, 322
caffe::NesterovSolver< Dtype >, 282	LayerSetUp, 323
caffe::Net	MaxBottomBlobs, 324
Backward, 288	MinBottomBlobs, 324
blob_loss_weights_, 289	Reshape, 324
bottom_vecs_, 289	caffe::RecurrentLayer< Dtype >, 318
ForwardFromTo, 288	caffe::ReductionLayer
learnable_param_ids_, 289	ExactNumBottomBlobs, 329
Reshape, 288	ExactNumTopBlobs, 329
ShareWeights, 288	LayerSetUp, 329
caffe::Net< Dtype >, 284	Reshape, 330
caffe::Net< Dtype >::Callback, 101	caffe::ReductionLayer< Dtype >, 325
caffe::NeuronLayer	caffe::ReshapeLayer
ExactNumBottomBlobs, 291	ExactNumBottomBlobs, 337
ExactNumTopBlobs, 291	ExactNumTopBlobs, 337
Reshape, 291	LayerSetUp, 338
caffe::NeuronLayer< Dtype >, 289	Reshape, 338
caffe::PReLULayer	caffe::ReshapeLayer< Dtype >, 335
Backward_cpu, 312	caffe::SGDSolver< Dtype >, 351
Forward cpu, 312	caffe::SPPLayer
LayerSetUp, 313	ExactNumBottomBlobs, 396
PReLULayer, 311	ExactNumTopBlobs, 397
Reshape, 313	LayerSetUp, 397
caffe::PReLULayer< Dtype >, 308	Reshape, 397
caffe::ParameterLayer	caffe::SPPLayer< Dtype >, 393
ExactNumBottomBlobs, 295	caffe::ScaleLayer
ExactNumTopBlobs, 295	ExactNumTopBlobs, 348
LayerSetUp, 295	Forward_cpu, 348
Reshape, 296	LayerSetUp, 349
•	•
caffe::ParameterLayer< Dtype >, 293	MaxBottomBlobs, 349
caffe::PoolingLayer	MinBottomBlobs, 350
ExactNumBottomBlobs, 299	Reshape, 350
LayerSetUp, 300	caffe::ScaleLayer< Dtype >, 345

caffe::SigmoidCrossEntropyLossLayer	Forward cpu, 409
Backward_cpu, 357	caffe::TanHLayer< Dtype >, 405
Forward_cpu, 357	caffe::ThresholdLayer
get_normalizer, 358	Forward_cpu, 412
LayerSetUp, 358	LayerSetUp, 413
Reshape, 359	ThresholdLayer, 412
caffe::SigmoidCrossEntropyLossLayer< Dtype >, 353	caffe::ThresholdLayer< Dtype >, 409
caffe::SigmoidLayer	caffe::TileLayer
Backward cpu, 362	ExactNumBottomBlobs, 416
Forward_cpu, 363	ExactNumTopBlobs, 416
caffe::SigmoidLayer< Dtype >, 359	Reshape, 417
caffe::SignalHandler, 363	caffe::TileLayer< Dtype >, 413
caffe::SilenceLayer	caffe::Timer, 418
ExactNumTopBlobs, 366	caffe::UniformFiller< Dtype >, 421
MinBottomBlobs, 366	caffe::WindowDataLayer
Reshape, 367	ExactNumBottomBlobs, 425
caffe::SilenceLayer< Dtype >, 364	ExactNumTopBlobs, 425
caffe::SliceLayer	caffe::WindowDataLayer< Dtype >, 422
ExactNumBottomBlobs, 370	caffe::XavierFiller< Dtype >, 426
,	caffe::db::Cursor, 130
LayerSetUp, 371	caffe::db::DB, 138
MinTopBlobs, 371	caffe::db::Transaction, 420
Reshape, 371	CanonicalAxisIndex
caffe::SliceLayer< Dtype >, 367	caffe::Blob, 91
caffe::SoftmaxLayer	CheckBlobCounts
ExactNumBottomBlobs, 376	caffe::Layer, 237
ExactNumTopBlobs, 377	ClipLayer
Reshape, 377	caffe::ClipLayer, 105
caffe::SoftmaxLayer< Dtype >, 373	ConvolutionLayer
caffe::SoftmaxWithLossLayer	caffe::ConvolutionLayer, 122
Backward_cpu, 382	CopyFrom
ExactNumTopBlobs, 383	caffe::Blob, 91
get_normalizer, 383	count
LayerSetUp, 383	caffe::Blob, 92
MaxTopBlobs, 384	caneblob, 92
MinTopBlobs, 384	DropoutLayer
Reshape, 384	caffe::DropoutLayer, 144
SoftmaxWithLossLayer, 382	oanonDropoutEayor, 111
caffe::SoftmaxWithLossLayer< Dtype >, 378	ELULayer
caffe::Solver< Dtype >, 385	caffe::ELULayer, 157
caffe::Solver< Dtype >::Callback, 102	EqualNumBottomTopBlobs
caffe::SolverAction, 29	caffe::BaseConvolutionLayer, 61
caffe::SolverRegisterer< Dtype >, 388	caffe::Layer, 237
caffe::SolverRegistry< Dtype >, 388	ExactNumBottomBlobs
caffe::SplitLayer	caffe::AbsValLayer, 35
ExactNumBottomBlobs, 392	caffe::AccuracyLayer, 40
MinTopBlobs, 392	caffe::ArgMaxLayer, 55
Reshape, 393	caffe::BatchNormLayer, 74
caffe::SplitLayer< Dtype >, 389	caffe::BatchReindexLayer, 79
caffe::SwishLayer	caffe::ContrastiveLossLayer, 117
Backward_cpu, 402	caffe::CropLayer, 128
Forward_cpu, 402	caffe::DataLayer, 133
LayerSetUp, 402	caffe::DummyDataLayer, 149
Reshape, 403	caffe::EmbedLayer, 161
SwishLayer, 401	caffe::FlattenLayer, 184
caffe::SwishLayer< Dtype >, 398	caffe::HDF5DataLayer, 191
caffe::SyncedMemory, 404	caffe::HDF5OutputLayer, 197
caffe::TanHLayer	caffe::Im2colLayer, 207
Backward_cpu, 408	caffe::ImageDataLayer, 212
Daunwaiu_upu, 400	caneimayeDalaLayet, 212

-the data weight and some 047	Famound and
caffe::InfogainLossLayer, 217	Forward_cpu
caffe::InnerProductLayer, 224	caffe::AbsValLayer, 35
caffe::InputLayer, 229	caffe::AccuracyLayer, 40
caffe::LRNLayer, 257	caffe::ArgMaxLayer, 55
caffe::LSTMUnitLayer, 265	caffe::BNLLLayer, 99
caffe::Layer, 237	caffe::BatchReindexLayer, 79
caffe::LossLayer, 251	caffe::ClipLayer, 106
caffe::MVNLayer, 280	caffe::ConcatLayer, 110
caffe::MemoryDataLayer, 269	caffe::ContrastiveLossLayer, 118
caffe::NeuronLayer, 291	caffe::DropoutLayer, 145
caffe::ParameterLayer, 295 caffe::PoolingLayer, 299	caffe::ELULayer, 158
caffe::ReductionLayer, 329	caffe::EuclideanLossLayer, 168
caffe::ReshapeLayer, 337	caffe::ExpLayer, 173
caffe::SPPLayer, 396	caffe::FilterLayer, 179
caffe::SliceLayer, 370	caffe::FlattenLayer, 185
caffe::SoftmaxLayer, 376	caffe::HingeLossLayer, 203
· ·	caffe::InfogainLossLayer, 217
caffe::SplitLayer, 392	caffe::LSTMUnitLayer, 265
caffe::TileLayer, 416	caffe::LogLayer, 248
caffe::WindowDataLayer, 425 ExactNumTopBlobs	caffe::MultinomialLogisticLossLayer, 276
caffe::AbsValLayer, 35	caffe::PReLULayer, 312
caffe::ArgMaxLayer, 55	caffe::PowerLayer, 307 caffe::ReLULayer, 334
caffe::BatchNormLayer, 74	caffe::RecurrentLayer, 322
caffe::BatchReindexLayer, 79	caffe::ScaleLayer, 348
caffe::BiasLayer, 83	caffe::SigmoidCrossEntropyLossLayer, 357
caffe::ConcatLayer, 110	caffe::SigmoidLayer, 363
caffe::CropLayer, 128	caffe::SwishLayer, 402
caffe::EltwiseLayer, 153	caffe::TanHLayer, 409
caffe::EmbedLayer, 162	caffe::ThresholdLayer, 412
caffe::FlattenLayer, 185	ForwardFromTo
caffe::HDF5OutputLayer, 198	caffe::Net, 288
caffe::Im2colLayer, 208	54115.11 TOI, 255
caffe::ImageDataLayer, 212	get_normalizer
caffe::InfogainLossLayer, 217	caffe::InfogainLossLayer, 219
caffe::InnerProductLayer, 225	caffe::SigmoidCrossEntropyLossLayer, 358
caffe::LRNLayer, 257	caffe::SoftmaxWithLossLayer, 383
caffe::LSTMUnitLayer, 265	GetFiller
caffe::Layer, 237	caffe, 28
caffe::LossLayer, 251	
caffe::MVNLayer, 281	InferBlobShape
caffe::MemoryDataLayer, 270	caffe::DataTransformer, 135
caffe::NeuronLayer, 291	
caffe::ParameterLayer, 295	Layer
caffe::RecurrentLayer, 322	caffe::Layer, 235
caffe::ReductionLayer, 329	layer_param_
caffe::ReshapeLayer, 337	caffe::Layer, 242
caffe::SPPLayer, 397	LayerSetUp
caffe::ScaleLayer, 348	caffe::AbsValLayer, 36
caffe::SilenceLayer, 366	caffe::AccuracyLayer, 41
caffe::SoftmaxLayer, 377	caffe::ArgMaxLayer, 56
caffe::SoftmaxWithLossLayer, 383	caffe::BaseConvolutionLayer, 61
caffe::TileLayer, 416	caffe::BaseDataLayer, 65 caffe::BasePrefetchingDataLayer, 69
caffe::WindowDataLayer, 425	caffe::BatchNormLayer, 74
ExpLayer 470	caffe::BiasLayer, 83
caffe::ExpLayer, 172	caffe::ConcatLayer, 111
Forward	caffe::ContrastiveLossLayer, 118
caffe::Layer, 238	caffe::CropLayer, 129
	1 7 7 -

caffe::DropoutLayer, 145	caffe::RecurrentLayer, 324
caffe::DummyDataLayer, 149	caffe::ScaleLayer, 350
caffe::EltwiseLayer, 153	caffe::SilenceLayer, 366
caffe::EmbedLayer, 162	MinTopBlobs
caffe::ExpLayer, 173	caffe::AccuracyLayer, 41
caffe::FilterLayer, 179	caffe::BaseConvolutionLayer, 62
caffe::HDF5DataLayer, 192	caffe::DataLayer, 133
caffe::HDF5OutputLayer, 198	caffe::DummyDataLayer, 150
caffe::Im2colLayer, 208	caffe::FilterLayer, 180
caffe::InfogainLossLayer, 219	caffe::HDF5DataLayer, 192
caffe::InnerProductLayer, 225	caffe::InfogainLossLayer, 220
caffe::InputLayer, 229	caffe::InputLayer, 230
caffe::LRNLayer, 257	caffe::Layer, 240
caffe::Layer, 238	caffe::PoolingLayer, 300
caffe::LogLayer, 248	caffe::SliceLayer, 371
caffe::LossLayer, 252	caffe::SoftmaxWithLossLayer, 384
caffe::PReLULayer, 313	caffe::SplitLayer, 392
caffe::ParameterLayer, 295	
caffe::PoolingLayer, 300	PReLULayer
caffe::PowerLayer, 308	caffe::PReLULayer, 311
caffe::PythonLayer, 317	param_propagate_down
caffe::RecurrentLayer, 323	caffe::Layer, 240
caffe::ReductionLayer, 329	param_propagate_down_
caffe::ReshapeLayer, 338	caffe::Layer, 242
caffe::SPPLayer, 397	phase_
caffe::ScaleLayer, 349	caffe::Layer, 242
caffe::SigmoidCrossEntropyLossLayer, 358	PowerLayer
caffe::SliceLayer, 371	caffe::PowerLayer, 306
caffe::SoftmaxWithLossLayer, 383	
caffe::SwishLayer, 402	Rand
caffe::ThresholdLayer, 413	caffe::DataTransformer, 136
learnable_param_ids_	ReLULayer
caffe::Net, 289	caffe::ReLULayer, 333
•	Reshape
LogLayer caffe::LogLayer, 247	caffe::AccuracyLayer, 41
• •	caffe::ArgMaxLayer, 56
loss_ caffe::Layer, 242	caffe::BaseConvolutionLayer, 62
CalleLayer, 242	caffe::BaseDataLayer, 65
MaxBottomBlobs	caffe::BatchNormLayer, 75
caffe::BiasLayer, 85	caffe::BatchReindexLayer, 80
caffe::InfogainLossLayer, 220	caffe::BiasLayer, 85
caffe::Layer, 239	caffe::Blob, 92
caffe::RecurrentLayer, 324	caffe::ConcatLayer, 111
caffe::ScaleLayer, 349	caffe::CropLayer, 129
MaxTopBlobs	caffe::DropoutLayer, 146
caffe::AccuracyLayer, 41	caffe::DummyDataLayer, 150
caffe::DataLayer, 133	caffe::EltwiseLayer, 154
caffe::InfogainLossLayer, 220	caffe::EmbedLayer, 162
caffe::Layer, 239	caffe::EuclideanLossLayer, 168
caffe::PoolingLayer, 300	caffe::FilterLayer, 180
caffe::SoftmaxWithLossLayer, 384	caffe::FlattenLayer, 185
MinBottomBlobs	caffe::HDF5DataLayer, 192
caffe::BaseConvolutionLayer, 61	caffe::HDF5OutputLayer, 198
caffe::BiasLayer, 85	caffe::Im2colLayer, 208
caffe::ConcatLayer, 111	caffe::InfogainLossLayer, 221
caffe::EltwiseLayer, 154	caffe::InnerProductLayer, 225
caffe::FilterLayer, 180	
	caffelnnutl.aver 220
<del>-</del>	caffe::InputLayer, 230
caffe::InfogainLossLayer, 220 caffe::Layer, 239	caffe::InputLayer, 230 caffe::LRNLayer, 258 caffe::LSTMUnitLayer, 266

```
caffe::Layer, 240
     caffe::LossLayer, 252
    caffe::MVNLayer, 281
    caffe::MultinomialLogisticLossLayer, 277
    caffe::Net, 288
    caffe::NeuronLayer, 291
    caffe::PReLULayer, 313
    caffe::ParameterLayer, 296
    caffe::PoolingLayer, 301
    caffe::PythonLayer, 317
    caffe::RecurrentLayer, 324
    caffe::ReductionLayer, 330
    caffe::ReshapeLayer, 338
    caffe::SPPLayer, 397
    caffe::ScaleLayer, 350
    caffe::SigmoidCrossEntropyLossLayer, 359
    caffe::SilenceLayer, 367
    caffe::SliceLayer, 371
    caffe::SoftmaxLayer, 377
    caffe::SoftmaxWithLossLayer, 384
    caffe::SplitLayer, 393
    caffe::SwishLayer, 403
    caffe::TileLayer, 417
SetLossWeights
     caffe::Layer, 241
SetUp
     caffe::Layer, 241
shape
    caffe::Blob, 93
ShareData
    caffe::Blob, 93
ShareDiff
    caffe::Blob, 93
ShareWeights
    caffe::Net, 288
SoftmaxWithLossLayer
    caffe::SoftmaxWithLossLayer, 382
StartInternalThread
     caffe::InternalThread, 232
StopInternalThread
    caffe::InternalThread, 232
SwishLayer
    caffe::SwishLayer, 401
ThresholdLayer
     caffe::ThresholdLayer, 412
Transform
    caffe::DataTransformer, 136, 137
```