JavaScript is disabled on your browser.

* [Overview](http://docs.google.com/overview-summary.html)
* [Package](http://docs.google.com/package-summary.html)
* Class
* [Tree](http://docs.google.com/package-tree.html)
* [Index](http://docs.google.com/index-all.html)
* [Help](http://docs.google.com/help-doc.html)
* [Prev Class](http://docs.google.com/org/opencv/core/Algorithm.html)
* [Next Class](http://docs.google.com/org/opencv/core/Core.MinMaxLocResult.html)
* [Frames](http://docs.google.com/index.html?org/opencv/core/Core.html)
* [No Frames](http://docs.google.com/Core.html)
* [All Classes](http://docs.google.com/allclasses-noframe.html)
* Summary:
* [Nested](#3znysh7) |
* [Field](#2et92p0) |
* [Constr](#tyjcwt) |
* [Method](#3dy6vkm)
* Detail:
* [Field](#4d34og8) |
* [Constr](#3abhhcj) |
* [Method](#49gfa85)

org.opencv.core

## Class Core

* java.lang.Object
  + org.opencv.core.Core
* public class Core  
  extends java.lang.Object

### Nested Class SummaryNested Classes

| Modifier and Type | Class and Description |
| --- | --- |
| static class | [**Core.MinMaxLocResult**](http://docs.google.com/org/opencv/core/Core.MinMaxLocResult.html) |

### Field SummaryFields

| Modifier and Type | Field and Description |
| --- | --- |
| static int | [**BadAlign**](http://docs.google.com/org/opencv/core/Core.html#BadAlign) |
| static int | [**BadAlphaChannel**](http://docs.google.com/org/opencv/core/Core.html#BadAlphaChannel) |
| static int | [**BadCallBack**](http://docs.google.com/org/opencv/core/Core.html#BadCallBack) |
| static int | [**BadCOI**](http://docs.google.com/org/opencv/core/Core.html#BadCOI) |
| static int | [**BadDataPtr**](http://docs.google.com/org/opencv/core/Core.html#BadDataPtr) |
| static int | [**BadDepth**](http://docs.google.com/org/opencv/core/Core.html#BadDepth) |
| static int | [**BadImageSize**](http://docs.google.com/org/opencv/core/Core.html#BadImageSize) |
| static int | [**BadModelOrChSeq**](http://docs.google.com/org/opencv/core/Core.html#BadModelOrChSeq) |
| static int | [**BadNumChannel1U**](http://docs.google.com/org/opencv/core/Core.html#BadNumChannel1U) |
| static int | [**BadNumChannels**](http://docs.google.com/org/opencv/core/Core.html#BadNumChannels) |
| static int | [**BadOffset**](http://docs.google.com/org/opencv/core/Core.html#BadOffset) |
| static int | [**BadOrder**](http://docs.google.com/org/opencv/core/Core.html#BadOrder) |
| static int | [**BadOrigin**](http://docs.google.com/org/opencv/core/Core.html#BadOrigin) |
| static int | [**BadROISize**](http://docs.google.com/org/opencv/core/Core.html#BadROISize) |
| static int | [**BadStep**](http://docs.google.com/org/opencv/core/Core.html#BadStep) |
| static int | [**BadTileSize**](http://docs.google.com/org/opencv/core/Core.html#BadTileSize) |
| static int | [**BORDER\_CONSTANT**](http://docs.google.com/org/opencv/core/Core.html#BORDER_CONSTANT) |
| static int | [**BORDER\_DEFAULT**](http://docs.google.com/org/opencv/core/Core.html#BORDER_DEFAULT) |
| static int | [**BORDER\_ISOLATED**](http://docs.google.com/org/opencv/core/Core.html#BORDER_ISOLATED) |
| static int | [**BORDER\_REFLECT**](http://docs.google.com/org/opencv/core/Core.html#BORDER_REFLECT) |
| static int | [**BORDER\_REFLECT\_101**](http://docs.google.com/org/opencv/core/Core.html#BORDER_REFLECT_101) |
| static int | [**BORDER\_REFLECT101**](http://docs.google.com/org/opencv/core/Core.html#BORDER_REFLECT101) |
| static int | [**BORDER\_REPLICATE**](http://docs.google.com/org/opencv/core/Core.html#BORDER_REPLICATE) |
| static int | [**BORDER\_TRANSPARENT**](http://docs.google.com/org/opencv/core/Core.html#BORDER_TRANSPARENT) |
| static int | [**BORDER\_WRAP**](http://docs.google.com/org/opencv/core/Core.html#BORDER_WRAP) |
| static int | [**CMP\_EQ**](http://docs.google.com/org/opencv/core/Core.html#CMP_EQ) |
| static int | [**CMP\_GE**](http://docs.google.com/org/opencv/core/Core.html#CMP_GE) |
| static int | [**CMP\_GT**](http://docs.google.com/org/opencv/core/Core.html#CMP_GT) |
| static int | [**CMP\_LE**](http://docs.google.com/org/opencv/core/Core.html#CMP_LE) |
| static int | [**CMP\_LT**](http://docs.google.com/org/opencv/core/Core.html#CMP_LT) |
| static int | [**CMP\_NE**](http://docs.google.com/org/opencv/core/Core.html#CMP_NE) |
| static int | [**COVAR\_COLS**](http://docs.google.com/org/opencv/core/Core.html#COVAR_COLS) |
| static int | [**COVAR\_NORMAL**](http://docs.google.com/org/opencv/core/Core.html#COVAR_NORMAL) |
| static int | [**COVAR\_ROWS**](http://docs.google.com/org/opencv/core/Core.html#COVAR_ROWS) |
| static int | [**COVAR\_SCALE**](http://docs.google.com/org/opencv/core/Core.html#COVAR_SCALE) |
| static int | [**COVAR\_SCRAMBLED**](http://docs.google.com/org/opencv/core/Core.html#COVAR_SCRAMBLED) |
| static int | [**COVAR\_USE\_AVG**](http://docs.google.com/org/opencv/core/Core.html#COVAR_USE_AVG) |
| static int | [**DCT\_INVERSE**](http://docs.google.com/org/opencv/core/Core.html#DCT_INVERSE) |
| static int | [**DCT\_ROWS**](http://docs.google.com/org/opencv/core/Core.html#DCT_ROWS) |
| static int | [**DECOMP\_CHOLESKY**](http://docs.google.com/org/opencv/core/Core.html#DECOMP_CHOLESKY) |
| static int | [**DECOMP\_EIG**](http://docs.google.com/org/opencv/core/Core.html#DECOMP_EIG) |
| static int | [**DECOMP\_LU**](http://docs.google.com/org/opencv/core/Core.html#DECOMP_LU) |
| static int | [**DECOMP\_NORMAL**](http://docs.google.com/org/opencv/core/Core.html#DECOMP_NORMAL) |
| static int | [**DECOMP\_QR**](http://docs.google.com/org/opencv/core/Core.html#DECOMP_QR) |
| static int | [**DECOMP\_SVD**](http://docs.google.com/org/opencv/core/Core.html#DECOMP_SVD) |
| static int | [**DFT\_COMPLEX\_INPUT**](http://docs.google.com/org/opencv/core/Core.html#DFT_COMPLEX_INPUT) |
| static int | [**DFT\_COMPLEX\_OUTPUT**](http://docs.google.com/org/opencv/core/Core.html#DFT_COMPLEX_OUTPUT) |
| static int | [**DFT\_INVERSE**](http://docs.google.com/org/opencv/core/Core.html#DFT_INVERSE) |
| static int | [**DFT\_REAL\_OUTPUT**](http://docs.google.com/org/opencv/core/Core.html#DFT_REAL_OUTPUT) |
| static int | [**DFT\_ROWS**](http://docs.google.com/org/opencv/core/Core.html#DFT_ROWS) |
| static int | [**DFT\_SCALE**](http://docs.google.com/org/opencv/core/Core.html#DFT_SCALE) |
| static int | [**FILLED**](http://docs.google.com/org/opencv/core/Core.html#FILLED) |
| static int | [**FONT\_HERSHEY\_COMPLEX**](http://docs.google.com/org/opencv/core/Core.html#FONT_HERSHEY_COMPLEX) |
| static int | [**FONT\_HERSHEY\_COMPLEX\_SMALL**](http://docs.google.com/org/opencv/core/Core.html#FONT_HERSHEY_COMPLEX_SMALL) |
| static int | [**FONT\_HERSHEY\_DUPLEX**](http://docs.google.com/org/opencv/core/Core.html#FONT_HERSHEY_DUPLEX) |
| static int | [**FONT\_HERSHEY\_PLAIN**](http://docs.google.com/org/opencv/core/Core.html#FONT_HERSHEY_PLAIN) |
| static int | [**FONT\_HERSHEY\_SCRIPT\_COMPLEX**](http://docs.google.com/org/opencv/core/Core.html#FONT_HERSHEY_SCRIPT_COMPLEX) |
| static int | [**FONT\_HERSHEY\_SCRIPT\_SIMPLEX**](http://docs.google.com/org/opencv/core/Core.html#FONT_HERSHEY_SCRIPT_SIMPLEX) |
| static int | [**FONT\_HERSHEY\_SIMPLEX**](http://docs.google.com/org/opencv/core/Core.html#FONT_HERSHEY_SIMPLEX) |
| static int | [**FONT\_HERSHEY\_TRIPLEX**](http://docs.google.com/org/opencv/core/Core.html#FONT_HERSHEY_TRIPLEX) |
| static int | [**FONT\_ITALIC**](http://docs.google.com/org/opencv/core/Core.html#FONT_ITALIC) |
| static int | [**Formatter\_FMT\_C**](http://docs.google.com/org/opencv/core/Core.html#Formatter_FMT_C) |
| static int | [**Formatter\_FMT\_CSV**](http://docs.google.com/org/opencv/core/Core.html#Formatter_FMT_CSV) |
| static int | [**Formatter\_FMT\_DEFAULT**](http://docs.google.com/org/opencv/core/Core.html#Formatter_FMT_DEFAULT) |
| static int | [**Formatter\_FMT\_MATLAB**](http://docs.google.com/org/opencv/core/Core.html#Formatter_FMT_MATLAB) |
| static int | [**Formatter\_FMT\_NUMPY**](http://docs.google.com/org/opencv/core/Core.html#Formatter_FMT_NUMPY) |
| static int | [**Formatter\_FMT\_PYTHON**](http://docs.google.com/org/opencv/core/Core.html#Formatter_FMT_PYTHON) |
| static int | [**GEMM\_1\_T**](http://docs.google.com/org/opencv/core/Core.html#GEMM_1_T) |
| static int | [**GEMM\_2\_T**](http://docs.google.com/org/opencv/core/Core.html#GEMM_2_T) |
| static int | [**GEMM\_3\_T**](http://docs.google.com/org/opencv/core/Core.html#GEMM_3_T) |
| static int | [**GpuApiCallError**](http://docs.google.com/org/opencv/core/Core.html#GpuApiCallError) |
| static int | [**GpuNotSupported**](http://docs.google.com/org/opencv/core/Core.html#GpuNotSupported) |
| static int | [**Hamming\_normType**](http://docs.google.com/org/opencv/core/Core.html#Hamming_normType) |
| static int | [**HeaderIsNull**](http://docs.google.com/org/opencv/core/Core.html#HeaderIsNull) |
| static int | [**KMEANS\_PP\_CENTERS**](http://docs.google.com/org/opencv/core/Core.html#KMEANS_PP_CENTERS) |
| static int | [**KMEANS\_RANDOM\_CENTERS**](http://docs.google.com/org/opencv/core/Core.html#KMEANS_RANDOM_CENTERS) |
| static int | [**KMEANS\_USE\_INITIAL\_LABELS**](http://docs.google.com/org/opencv/core/Core.html#KMEANS_USE_INITIAL_LABELS) |
| static int | [**LINE\_4**](http://docs.google.com/org/opencv/core/Core.html#LINE_4) |
| static int | [**LINE\_8**](http://docs.google.com/org/opencv/core/Core.html#LINE_8) |
| static int | [**LINE\_AA**](http://docs.google.com/org/opencv/core/Core.html#LINE_AA) |
| static int | [**MaskIsTiled**](http://docs.google.com/org/opencv/core/Core.html#MaskIsTiled) |
| static java.lang.String | [**NATIVE\_LIBRARY\_NAME**](http://docs.google.com/org/opencv/core/Core.html#NATIVE_LIBRARY_NAME) |
| static int | [**NORM\_HAMMING**](http://docs.google.com/org/opencv/core/Core.html#NORM_HAMMING) |
| static int | [**NORM\_HAMMING2**](http://docs.google.com/org/opencv/core/Core.html#NORM_HAMMING2) |
| static int | [**NORM\_INF**](http://docs.google.com/org/opencv/core/Core.html#NORM_INF) |
| static int | [**NORM\_L1**](http://docs.google.com/org/opencv/core/Core.html#NORM_L1) |
| static int | [**NORM\_L2**](http://docs.google.com/org/opencv/core/Core.html#NORM_L2) |
| static int | [**NORM\_L2SQR**](http://docs.google.com/org/opencv/core/Core.html#NORM_L2SQR) |
| static int | [**NORM\_MINMAX**](http://docs.google.com/org/opencv/core/Core.html#NORM_MINMAX) |
| static int | [**NORM\_RELATIVE**](http://docs.google.com/org/opencv/core/Core.html#NORM_RELATIVE) |
| static int | [**NORM\_TYPE\_MASK**](http://docs.google.com/org/opencv/core/Core.html#NORM_TYPE_MASK) |
| static int | [**OpenCLApiCallError**](http://docs.google.com/org/opencv/core/Core.html#OpenCLApiCallError) |
| static int | [**OpenCLDoubleNotSupported**](http://docs.google.com/org/opencv/core/Core.html#OpenCLDoubleNotSupported) |
| static int | [**OpenCLInitError**](http://docs.google.com/org/opencv/core/Core.html#OpenCLInitError) |
| static int | [**OpenCLNoAMDBlasFft**](http://docs.google.com/org/opencv/core/Core.html#OpenCLNoAMDBlasFft) |
| static int | [**OpenGlApiCallError**](http://docs.google.com/org/opencv/core/Core.html#OpenGlApiCallError) |
| static int | [**OpenGlNotSupported**](http://docs.google.com/org/opencv/core/Core.html#OpenGlNotSupported) |
| static int | [**Param\_ALGORITHM**](http://docs.google.com/org/opencv/core/Core.html#Param_ALGORITHM) |
| static int | [**Param\_BOOLEAN**](http://docs.google.com/org/opencv/core/Core.html#Param_BOOLEAN) |
| static int | [**Param\_FLOAT**](http://docs.google.com/org/opencv/core/Core.html#Param_FLOAT) |
| static int | [**Param\_INT**](http://docs.google.com/org/opencv/core/Core.html#Param_INT) |
| static int | [**Param\_MAT**](http://docs.google.com/org/opencv/core/Core.html#Param_MAT) |
| static int | [**Param\_MAT\_VECTOR**](http://docs.google.com/org/opencv/core/Core.html#Param_MAT_VECTOR) |
| static int | [**Param\_REAL**](http://docs.google.com/org/opencv/core/Core.html#Param_REAL) |
| static int | [**Param\_SCALAR**](http://docs.google.com/org/opencv/core/Core.html#Param_SCALAR) |
| static int | [**Param\_STRING**](http://docs.google.com/org/opencv/core/Core.html#Param_STRING) |
| static int | [**Param\_UCHAR**](http://docs.google.com/org/opencv/core/Core.html#Param_UCHAR) |
| static int | [**Param\_UINT64**](http://docs.google.com/org/opencv/core/Core.html#Param_UINT64) |
| static int | [**Param\_UNSIGNED\_INT**](http://docs.google.com/org/opencv/core/Core.html#Param_UNSIGNED_INT) |
| static int | [**PCA\_DATA\_AS\_COL**](http://docs.google.com/org/opencv/core/Core.html#PCA_DATA_AS_COL) |
| static int | [**PCA\_DATA\_AS\_ROW**](http://docs.google.com/org/opencv/core/Core.html#PCA_DATA_AS_ROW) |
| static int | [**PCA\_USE\_AVG**](http://docs.google.com/org/opencv/core/Core.html#PCA_USE_AVG) |
| static int | [**REDUCE\_AVG**](http://docs.google.com/org/opencv/core/Core.html#REDUCE_AVG) |
| static int | [**REDUCE\_MAX**](http://docs.google.com/org/opencv/core/Core.html#REDUCE_MAX) |
| static int | [**REDUCE\_MIN**](http://docs.google.com/org/opencv/core/Core.html#REDUCE_MIN) |
| static int | [**REDUCE\_SUM**](http://docs.google.com/org/opencv/core/Core.html#REDUCE_SUM) |
| static int | [**RNG\_NORMAL**](http://docs.google.com/org/opencv/core/Core.html#RNG_NORMAL) |
| static int | [**RNG\_UNIFORM**](http://docs.google.com/org/opencv/core/Core.html#RNG_UNIFORM) |
| static int | [**ROTATE\_180**](http://docs.google.com/org/opencv/core/Core.html#ROTATE_180) |
| static int | [**ROTATE\_90\_CLOCKWISE**](http://docs.google.com/org/opencv/core/Core.html#ROTATE_90_CLOCKWISE) |
| static int | [**ROTATE\_90\_COUNTERCLOCKWISE**](http://docs.google.com/org/opencv/core/Core.html#ROTATE_90_COUNTERCLOCKWISE) |
| static int | [**SORT\_ASCENDING**](http://docs.google.com/org/opencv/core/Core.html#SORT_ASCENDING) |
| static int | [**SORT\_DESCENDING**](http://docs.google.com/org/opencv/core/Core.html#SORT_DESCENDING) |
| static int | [**SORT\_EVERY\_COLUMN**](http://docs.google.com/org/opencv/core/Core.html#SORT_EVERY_COLUMN) |
| static int | [**SORT\_EVERY\_ROW**](http://docs.google.com/org/opencv/core/Core.html#SORT_EVERY_ROW) |
| static int | [**StsAssert**](http://docs.google.com/org/opencv/core/Core.html#StsAssert) |
| static int | [**StsAutoTrace**](http://docs.google.com/org/opencv/core/Core.html#StsAutoTrace) |
| static int | [**StsBackTrace**](http://docs.google.com/org/opencv/core/Core.html#StsBackTrace) |
| static int | [**StsBadArg**](http://docs.google.com/org/opencv/core/Core.html#StsBadArg) |
| static int | [**StsBadFlag**](http://docs.google.com/org/opencv/core/Core.html#StsBadFlag) |
| static int | [**StsBadFunc**](http://docs.google.com/org/opencv/core/Core.html#StsBadFunc) |
| static int | [**StsBadMask**](http://docs.google.com/org/opencv/core/Core.html#StsBadMask) |
| static int | [**StsBadMemBlock**](http://docs.google.com/org/opencv/core/Core.html#StsBadMemBlock) |
| static int | [**StsBadPoint**](http://docs.google.com/org/opencv/core/Core.html#StsBadPoint) |
| static int | [**StsBadSize**](http://docs.google.com/org/opencv/core/Core.html#StsBadSize) |
| static int | [**StsDivByZero**](http://docs.google.com/org/opencv/core/Core.html#StsDivByZero) |
| static int | [**StsError**](http://docs.google.com/org/opencv/core/Core.html#StsError) |
| static int | [**StsFilterOffsetErr**](http://docs.google.com/org/opencv/core/Core.html#StsFilterOffsetErr) |
| static int | [**StsFilterStructContentErr**](http://docs.google.com/org/opencv/core/Core.html#StsFilterStructContentErr) |
| static int | [**StsInplaceNotSupported**](http://docs.google.com/org/opencv/core/Core.html#StsInplaceNotSupported) |
| static int | [**StsInternal**](http://docs.google.com/org/opencv/core/Core.html#StsInternal) |
| static int | [**StsKernelStructContentErr**](http://docs.google.com/org/opencv/core/Core.html#StsKernelStructContentErr) |
| static int | [**StsNoConv**](http://docs.google.com/org/opencv/core/Core.html#StsNoConv) |
| static int | [**StsNoMem**](http://docs.google.com/org/opencv/core/Core.html#StsNoMem) |
| static int | [**StsNotImplemented**](http://docs.google.com/org/opencv/core/Core.html#StsNotImplemented) |
| static int | [**StsNullPtr**](http://docs.google.com/org/opencv/core/Core.html#StsNullPtr) |
| static int | [**StsObjectNotFound**](http://docs.google.com/org/opencv/core/Core.html#StsObjectNotFound) |
| static int | [**StsOk**](http://docs.google.com/org/opencv/core/Core.html#StsOk) |
| static int | [**StsOutOfRange**](http://docs.google.com/org/opencv/core/Core.html#StsOutOfRange) |
| static int | [**StsParseError**](http://docs.google.com/org/opencv/core/Core.html#StsParseError) |
| static int | [**StsUnmatchedFormats**](http://docs.google.com/org/opencv/core/Core.html#StsUnmatchedFormats) |
| static int | [**StsUnmatchedSizes**](http://docs.google.com/org/opencv/core/Core.html#StsUnmatchedSizes) |
| static int | [**StsUnsupportedFormat**](http://docs.google.com/org/opencv/core/Core.html#StsUnsupportedFormat) |
| static int | [**StsVecLengthErr**](http://docs.google.com/org/opencv/core/Core.html#StsVecLengthErr) |
| static int | [**SVD\_FULL\_UV**](http://docs.google.com/org/opencv/core/Core.html#SVD_FULL_UV) |
| static int | [**SVD\_MODIFY\_A**](http://docs.google.com/org/opencv/core/Core.html#SVD_MODIFY_A) |
| static int | [**SVD\_NO\_UV**](http://docs.google.com/org/opencv/core/Core.html#SVD_NO_UV) |
| static java.lang.String | [**VERSION**](http://docs.google.com/org/opencv/core/Core.html#VERSION) |
| static int | [**VERSION\_MAJOR**](http://docs.google.com/org/opencv/core/Core.html#VERSION_MAJOR) |
| static int | [**VERSION\_MINOR**](http://docs.google.com/org/opencv/core/Core.html#VERSION_MINOR) |
| static int | [**VERSION\_REVISION**](http://docs.google.com/org/opencv/core/Core.html#VERSION_REVISION) |
| static java.lang.String | [**VERSION\_STATUS**](http://docs.google.com/org/opencv/core/Core.html#VERSION_STATUS) |

### Constructor SummaryConstructors

| Constructor and Description |
| --- |
| [**Core**](http://docs.google.com/org/opencv/core/Core.html#Core())() |

### Method SummaryMethods

| Modifier and Type | Method and Description |
| --- | --- |
| static void | [**absdiff**](http://docs.google.com/org/opencv/core/Core.html#absdiff(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Calculates the per-element absolute difference between two arrays or between an array and a scalar. |
| static void | [**absdiff**](http://docs.google.com/org/opencv/core/Core.html#absdiff(org.opencv.core.Mat,%20org.opencv.core.Scalar,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) |
| static void | [**add**](http://docs.google.com/org/opencv/core/Core.html#add(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Calculates the per-element sum of two arrays or an array and a scalar. |
| static void | [**add**](http://docs.google.com/org/opencv/core/Core.html#add(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) Calculates the per-element sum of two arrays or an array and a scalar. |
| static void | [**add**](http://docs.google.com/org/opencv/core/Core.html#add(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask, int dtype) Calculates the per-element sum of two arrays or an array and a scalar. |
| static void | [**add**](http://docs.google.com/org/opencv/core/Core.html#add(org.opencv.core.Mat,%20org.opencv.core.Scalar,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) |
| static void | [**add**](http://docs.google.com/org/opencv/core/Core.html#add(org.opencv.core.Mat,%20org.opencv.core.Scalar,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) |
| static void | [**add**](http://docs.google.com/org/opencv/core/Core.html#add(org.opencv.core.Mat,%20org.opencv.core.Scalar,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask, int dtype) |
| static void | [**addSamplesDataSearchPath**](http://docs.google.com/org/opencv/core/Core.html#addSamplesDataSearchPath(java.lang.String))(java.lang.String path) Override search data path by adding new search location Use this only to override default behavior Passed paths are used in LIFO order. |
| static void | [**addSamplesDataSearchSubDirectory**](http://docs.google.com/org/opencv/core/Core.html#addSamplesDataSearchSubDirectory(java.lang.String))(java.lang.String subdir) Append samples search data sub directory General usage is to add OpenCV modules name (<opencv\_contrib>/modules/<name>/samples/data -> <name>/samples/data + modules/<name>/samples/data). |
| static void | [**addWeighted**](http://docs.google.com/org/opencv/core/Core.html#addWeighted(org.opencv.core.Mat,%20double,%20org.opencv.core.Mat,%20double,%20double,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, double alpha, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, double beta, double gamma, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Calculates the weighted sum of two arrays. |
| static void | [**addWeighted**](http://docs.google.com/org/opencv/core/Core.html#addWeighted(org.opencv.core.Mat,%20double,%20org.opencv.core.Mat,%20double,%20double,%20org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, double alpha, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, double beta, double gamma, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int dtype) Calculates the weighted sum of two arrays. |
| static void | [**batchDistance**](http://docs.google.com/org/opencv/core/Core.html#batchDistance(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dist, int dtype, [Mat](http://docs.google.com/org/opencv/core/Mat.html) nidx) naive nearest neighbor finder see http://en.wikipedia.org/wiki/Nearest\_neighbor\_search TODO: document |
| static void | [**batchDistance**](http://docs.google.com/org/opencv/core/Core.html#batchDistance(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dist, int dtype, [Mat](http://docs.google.com/org/opencv/core/Mat.html) nidx, int normType) naive nearest neighbor finder see http://en.wikipedia.org/wiki/Nearest\_neighbor\_search TODO: document |
| static void | [**batchDistance**](http://docs.google.com/org/opencv/core/Core.html#batchDistance(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20org.opencv.core.Mat,%20int,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dist, int dtype, [Mat](http://docs.google.com/org/opencv/core/Mat.html) nidx, int normType, int K) naive nearest neighbor finder see http://en.wikipedia.org/wiki/Nearest\_neighbor\_search TODO: document |
| static void | [**batchDistance**](http://docs.google.com/org/opencv/core/Core.html#batchDistance(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20org.opencv.core.Mat,%20int,%20int,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dist, int dtype, [Mat](http://docs.google.com/org/opencv/core/Mat.html) nidx, int normType, int K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) naive nearest neighbor finder see http://en.wikipedia.org/wiki/Nearest\_neighbor\_search TODO: document |
| static void | [**batchDistance**](http://docs.google.com/org/opencv/core/Core.html#batchDistance(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20org.opencv.core.Mat,%20int,%20int,%20org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dist, int dtype, [Mat](http://docs.google.com/org/opencv/core/Mat.html) nidx, int normType, int K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask, int update) naive nearest neighbor finder see http://en.wikipedia.org/wiki/Nearest\_neighbor\_search TODO: document |
| static void | [**batchDistance**](http://docs.google.com/org/opencv/core/Core.html#batchDistance(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20org.opencv.core.Mat,%20int,%20int,%20org.opencv.core.Mat,%20int,%20boolean))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dist, int dtype, [Mat](http://docs.google.com/org/opencv/core/Mat.html) nidx, int normType, int K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask, int update, boolean crosscheck) naive nearest neighbor finder see http://en.wikipedia.org/wiki/Nearest\_neighbor\_search TODO: document |
| static void | [**bitwise\_and**](http://docs.google.com/org/opencv/core/Core.html#bitwise_and(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) computes bitwise conjunction of the two arrays (dst = src1 & src2) Calculates the per-element bit-wise conjunction of two arrays or an array and a scalar. |
| static void | [**bitwise\_and**](http://docs.google.com/org/opencv/core/Core.html#bitwise_and(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) computes bitwise conjunction of the two arrays (dst = src1 & src2) Calculates the per-element bit-wise conjunction of two arrays or an array and a scalar. |
| static void | [**bitwise\_not**](http://docs.google.com/org/opencv/core/Core.html#bitwise_not(org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Inverts every bit of an array. |
| static void | [**bitwise\_not**](http://docs.google.com/org/opencv/core/Core.html#bitwise_not(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) Inverts every bit of an array. |
| static void | [**bitwise\_or**](http://docs.google.com/org/opencv/core/Core.html#bitwise_or(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Calculates the per-element bit-wise disjunction of two arrays or an array and a scalar. |
| static void | [**bitwise\_or**](http://docs.google.com/org/opencv/core/Core.html#bitwise_or(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) Calculates the per-element bit-wise disjunction of two arrays or an array and a scalar. |
| static void | [**bitwise\_xor**](http://docs.google.com/org/opencv/core/Core.html#bitwise_xor(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Calculates the per-element bit-wise "exclusive or" operation on two arrays or an array and a scalar. |
| static void | [**bitwise\_xor**](http://docs.google.com/org/opencv/core/Core.html#bitwise_xor(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) Calculates the per-element bit-wise "exclusive or" operation on two arrays or an array and a scalar. |
| static int | [**borderInterpolate**](http://docs.google.com/org/opencv/core/Core.html#borderInterpolate(int,%20int,%20int))(int p, int len, int borderType) Computes the source location of an extrapolated pixel. |
| static void | [**calcCovarMatrix**](http://docs.google.com/org/opencv/core/Core.html#calcCovarMatrix(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) samples, [Mat](http://docs.google.com/org/opencv/core/Mat.html) covar, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mean, int flags) **Note:** use #COVAR\_ROWS or #COVAR\_COLS flag |
| static void | [**calcCovarMatrix**](http://docs.google.com/org/opencv/core/Core.html#calcCovarMatrix(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) samples, [Mat](http://docs.google.com/org/opencv/core/Mat.html) covar, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mean, int flags, int ctype) **Note:** use #COVAR\_ROWS or #COVAR\_COLS flag |
| static void | [**cartToPolar**](http://docs.google.com/org/opencv/core/Core.html#cartToPolar(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) x, [Mat](http://docs.google.com/org/opencv/core/Mat.html) y, [Mat](http://docs.google.com/org/opencv/core/Mat.html) magnitude, [Mat](http://docs.google.com/org/opencv/core/Mat.html) angle) Calculates the magnitude and angle of 2D vectors. |
| static void | [**cartToPolar**](http://docs.google.com/org/opencv/core/Core.html#cartToPolar(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20boolean))([Mat](http://docs.google.com/org/opencv/core/Mat.html) x, [Mat](http://docs.google.com/org/opencv/core/Mat.html) y, [Mat](http://docs.google.com/org/opencv/core/Mat.html) magnitude, [Mat](http://docs.google.com/org/opencv/core/Mat.html) angle, boolean angleInDegrees) Calculates the magnitude and angle of 2D vectors. |
| static boolean | [**checkRange**](http://docs.google.com/org/opencv/core/Core.html#checkRange(org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) a) Checks every element of an input array for invalid values. |
| static boolean | [**checkRange**](http://docs.google.com/org/opencv/core/Core.html#checkRange(org.opencv.core.Mat,%20boolean))([Mat](http://docs.google.com/org/opencv/core/Mat.html) a, boolean quiet) Checks every element of an input array for invalid values. |
| static boolean | [**checkRange**](http://docs.google.com/org/opencv/core/Core.html#checkRange(org.opencv.core.Mat,%20boolean,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) a, boolean quiet, double minVal) Checks every element of an input array for invalid values. |
| static boolean | [**checkRange**](http://docs.google.com/org/opencv/core/Core.html#checkRange(org.opencv.core.Mat,%20boolean,%20double,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) a, boolean quiet, double minVal, double maxVal) Checks every element of an input array for invalid values. |
| static void | [**compare**](http://docs.google.com/org/opencv/core/Core.html#compare(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int cmpop) Performs the per-element comparison of two arrays or an array and scalar value. |
| static void | [**compare**](http://docs.google.com/org/opencv/core/Core.html#compare(org.opencv.core.Mat,%20org.opencv.core.Scalar,%20org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int cmpop) |
| static void | [**completeSymm**](http://docs.google.com/org/opencv/core/Core.html#completeSymm(org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) m) Copies the lower or the upper half of a square matrix to its another half. |
| static void | [**completeSymm**](http://docs.google.com/org/opencv/core/Core.html#completeSymm(org.opencv.core.Mat,%20boolean))([Mat](http://docs.google.com/org/opencv/core/Mat.html) m, boolean lowerToUpper) Copies the lower or the upper half of a square matrix to its another half. |
| static void | [**convertFp16**](http://docs.google.com/org/opencv/core/Core.html#convertFp16(org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Converts an array to half precision floating number. |
| static void | [**convertScaleAbs**](http://docs.google.com/org/opencv/core/Core.html#convertScaleAbs(org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Scales, calculates absolute values, and converts the result to 8-bit. |
| static void | [**convertScaleAbs**](http://docs.google.com/org/opencv/core/Core.html#convertScaleAbs(org.opencv.core.Mat,%20org.opencv.core.Mat,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double alpha) Scales, calculates absolute values, and converts the result to 8-bit. |
| static void | [**convertScaleAbs**](http://docs.google.com/org/opencv/core/Core.html#convertScaleAbs(org.opencv.core.Mat,%20org.opencv.core.Mat,%20double,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double alpha, double beta) Scales, calculates absolute values, and converts the result to 8-bit. |
| static void | [**copyMakeBorder**](http://docs.google.com/org/opencv/core/Core.html#copyMakeBorder(org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20int,%20int,%20int,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int top, int bottom, int left, int right, int borderType) Forms a border around an image. |
| static void | [**copyMakeBorder**](http://docs.google.com/org/opencv/core/Core.html#copyMakeBorder(org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20int,%20int,%20int,%20int,%20org.opencv.core.Scalar))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int top, int bottom, int left, int right, int borderType, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) value) Forms a border around an image. |
| static int | [**countNonZero**](http://docs.google.com/org/opencv/core/Core.html#countNonZero(org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src) Counts non-zero array elements. |
| static float | [**cubeRoot**](http://docs.google.com/org/opencv/core/Core.html#cubeRoot(float))(float val) Computes the cube root of an argument. |
| static void | [**dct**](http://docs.google.com/org/opencv/core/Core.html#dct(org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Performs a forward or inverse discrete Cosine transform of 1D or 2D array. |
| static void | [**dct**](http://docs.google.com/org/opencv/core/Core.html#dct(org.opencv.core.Mat,%20org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int flags) Performs a forward or inverse discrete Cosine transform of 1D or 2D array. |
| static double | [**determinant**](http://docs.google.com/org/opencv/core/Core.html#determinant(org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) mtx) Returns the determinant of a square floating-point matrix. |
| static void | [**dft**](http://docs.google.com/org/opencv/core/Core.html#dft(org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Performs a forward or inverse Discrete Fourier transform of a 1D or 2D floating-point array. |
| static void | [**dft**](http://docs.google.com/org/opencv/core/Core.html#dft(org.opencv.core.Mat,%20org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int flags) Performs a forward or inverse Discrete Fourier transform of a 1D or 2D floating-point array. |
| static void | [**dft**](http://docs.google.com/org/opencv/core/Core.html#dft(org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int flags, int nonzeroRows) Performs a forward or inverse Discrete Fourier transform of a 1D or 2D floating-point array. |
| static void | [**divide**](http://docs.google.com/org/opencv/core/Core.html#divide(double,%20org.opencv.core.Mat,%20org.opencv.core.Mat))(double scale, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) |
| static void | [**divide**](http://docs.google.com/org/opencv/core/Core.html#divide(double,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int))(double scale, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int dtype) |
| static void | [**divide**](http://docs.google.com/org/opencv/core/Core.html#divide(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Performs per-element division of two arrays or a scalar by an array. |
| static void | [**divide**](http://docs.google.com/org/opencv/core/Core.html#divide(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double scale) Performs per-element division of two arrays or a scalar by an array. |
| static void | [**divide**](http://docs.google.com/org/opencv/core/Core.html#divide(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20double,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double scale, int dtype) Performs per-element division of two arrays or a scalar by an array. |
| static void | [**divide**](http://docs.google.com/org/opencv/core/Core.html#divide(org.opencv.core.Mat,%20org.opencv.core.Scalar,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) |
| static void | [**divide**](http://docs.google.com/org/opencv/core/Core.html#divide(org.opencv.core.Mat,%20org.opencv.core.Scalar,%20org.opencv.core.Mat,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double scale) |
| static void | [**divide**](http://docs.google.com/org/opencv/core/Core.html#divide(org.opencv.core.Mat,%20org.opencv.core.Scalar,%20org.opencv.core.Mat,%20double,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double scale, int dtype) |
| static boolean | [**eigen**](http://docs.google.com/org/opencv/core/Core.html#eigen(org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) eigenvalues) Calculates eigenvalues and eigenvectors of a symmetric matrix. |
| static boolean | [**eigen**](http://docs.google.com/org/opencv/core/Core.html#eigen(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) eigenvalues, [Mat](http://docs.google.com/org/opencv/core/Mat.html) eigenvectors) Calculates eigenvalues and eigenvectors of a symmetric matrix. |
| static void | [**eigenNonSymmetric**](http://docs.google.com/org/opencv/core/Core.html#eigenNonSymmetric(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) eigenvalues, [Mat](http://docs.google.com/org/opencv/core/Mat.html) eigenvectors) Calculates eigenvalues and eigenvectors of a non-symmetric matrix (real eigenvalues only). |
| static void | [**exp**](http://docs.google.com/org/opencv/core/Core.html#exp(org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Calculates the exponent of every array element. |
| static void | [**extractChannel**](http://docs.google.com/org/opencv/core/Core.html#extractChannel(org.opencv.core.Mat,%20org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int coi) Extracts a single channel from src (coi is 0-based index) |
| static float | [**fastAtan2**](http://docs.google.com/org/opencv/core/Core.html#fastAtan2(float,%20float))(float y, float x) Calculates the angle of a 2D vector in degrees. |
| static java.lang.String | [**findFile**](http://docs.google.com/org/opencv/core/Core.html#findFile(java.lang.String))(java.lang.String relative\_path) Try to find requested data file Search directories: 1. |
| static java.lang.String | [**findFile**](http://docs.google.com/org/opencv/core/Core.html#findFile(java.lang.String,%20boolean))(java.lang.String relative\_path, boolean required) Try to find requested data file Search directories: 1. |
| static java.lang.String | [**findFile**](http://docs.google.com/org/opencv/core/Core.html#findFile(java.lang.String,%20boolean,%20boolean))(java.lang.String relative\_path, boolean required, boolean silentMode) Try to find requested data file Search directories: 1. |
| static java.lang.String | [**findFileOrKeep**](http://docs.google.com/org/opencv/core/Core.html#findFileOrKeep(java.lang.String))(java.lang.String relative\_path) |
| static java.lang.String | [**findFileOrKeep**](http://docs.google.com/org/opencv/core/Core.html#findFileOrKeep(java.lang.String,%20boolean))(java.lang.String relative\_path, boolean silentMode) |
| static void | [**findNonZero**](http://docs.google.com/org/opencv/core/Core.html#findNonZero(org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) idx) Returns the list of locations of non-zero pixels Given a binary matrix (likely returned from an operation such as threshold(), compare(), >, ==, etc, return all of the non-zero indices as a cv::Mat or std::vector<cv::Point> (x,y) For example: cv::Mat binaryImage; // input, binary image cv::Mat locations; // output, locations of non-zero pixels cv::findNonZero(binaryImage, locations); // access pixel coordinates Point pnt = locations.at<Point>(i); or cv::Mat binaryImage; // input, binary image vector<Point> locations; // output, locations of non-zero pixels cv::findNonZero(binaryImage, locations); // access pixel coordinates Point pnt = locations[i]; |
| static void | [**flip**](http://docs.google.com/org/opencv/core/Core.html#flip(org.opencv.core.Mat,%20org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int flipCode) Flips a 2D array around vertical, horizontal, or both axes. |
| static void | [**gemm**](http://docs.google.com/org/opencv/core/Core.html#gemm(org.opencv.core.Mat,%20org.opencv.core.Mat,%20double,%20org.opencv.core.Mat,%20double,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, double alpha, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src3, double beta, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Performs generalized matrix multiplication. |
| static void | [**gemm**](http://docs.google.com/org/opencv/core/Core.html#gemm(org.opencv.core.Mat,%20org.opencv.core.Mat,%20double,%20org.opencv.core.Mat,%20double,%20org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, double alpha, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src3, double beta, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int flags) Performs generalized matrix multiplication. |
| static java.lang.String | [**getBuildInformation**](http://docs.google.com/org/opencv/core/Core.html#getBuildInformation())() Returns full configuration time cmake output. |
| static long | [**getCPUTickCount**](http://docs.google.com/org/opencv/core/Core.html#getCPUTickCount())() Returns the number of CPU ticks. |
| static java.lang.String | [**getHardwareFeatureName**](http://docs.google.com/org/opencv/core/Core.html#getHardwareFeatureName(int))(int feature) Returns feature name by ID Returns empty string if feature is not defined |
| static java.lang.String | [**getIppVersion**](http://docs.google.com/org/opencv/core/Core.html#getIppVersion())() |
| static int | [**getNumberOfCPUs**](http://docs.google.com/org/opencv/core/Core.html#getNumberOfCPUs())() Returns the number of logical CPUs available for the process. |
| static int | [**getNumThreads**](http://docs.google.com/org/opencv/core/Core.html#getNumThreads())() Returns the number of threads used by OpenCV for parallel regions. |
| static int | [**getOptimalDFTSize**](http://docs.google.com/org/opencv/core/Core.html#getOptimalDFTSize(int))(int vecsize) Returns the optimal DFT size for a given vector size. |
| static long | [**getTickCount**](http://docs.google.com/org/opencv/core/Core.html#getTickCount())() Returns the number of ticks. |
| static double | [**getTickFrequency**](http://docs.google.com/org/opencv/core/Core.html#getTickFrequency())() Returns the number of ticks per second. |
| static int | [**getVersionMajor**](http://docs.google.com/org/opencv/core/Core.html#getVersionMajor())() Returns major library version |
| static int | [**getVersionMinor**](http://docs.google.com/org/opencv/core/Core.html#getVersionMinor())() Returns minor library version |
| static int | [**getVersionRevision**](http://docs.google.com/org/opencv/core/Core.html#getVersionRevision())() Returns revision field of the library version |
| static java.lang.String | [**getVersionString**](http://docs.google.com/org/opencv/core/Core.html#getVersionString())() Returns library version string For example "3.4.1-dev". |
| static void | [**hconcat**](http://docs.google.com/org/opencv/core/Core.html#hconcat(java.util.List,%20org.opencv.core.Mat))(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) std::vector<cv::Mat> matrices = { cv::Mat(4, 1, CV\_8UC1, cv::Scalar(1)), cv::Mat(4, 1, CV\_8UC1, cv::Scalar(2)), cv::Mat(4, 1, CV\_8UC1, cv::Scalar(3)),}; cv::Mat out; cv::hconcat( matrices, out ); //out: //[1, 2, 3; // 1, 2, 3; // 1, 2, 3; // 1, 2, 3] |
| static void | [**idct**](http://docs.google.com/org/opencv/core/Core.html#idct(org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Calculates the inverse Discrete Cosine Transform of a 1D or 2D array. |
| static void | [**idct**](http://docs.google.com/org/opencv/core/Core.html#idct(org.opencv.core.Mat,%20org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int flags) Calculates the inverse Discrete Cosine Transform of a 1D or 2D array. |
| static void | [**idft**](http://docs.google.com/org/opencv/core/Core.html#idft(org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Calculates the inverse Discrete Fourier Transform of a 1D or 2D array. |
| static void | [**idft**](http://docs.google.com/org/opencv/core/Core.html#idft(org.opencv.core.Mat,%20org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int flags) Calculates the inverse Discrete Fourier Transform of a 1D or 2D array. |
| static void | [**idft**](http://docs.google.com/org/opencv/core/Core.html#idft(org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int flags, int nonzeroRows) Calculates the inverse Discrete Fourier Transform of a 1D or 2D array. |
| static void | [**inRange**](http://docs.google.com/org/opencv/core/Core.html#inRange(org.opencv.core.Mat,%20org.opencv.core.Scalar,%20org.opencv.core.Scalar,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) lowerb, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) upperb, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Checks if array elements lie between the elements of two other arrays. |
| static void | [**insertChannel**](http://docs.google.com/org/opencv/core/Core.html#insertChannel(org.opencv.core.Mat,%20org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int coi) Inserts a single channel to dst (coi is 0-based index) |
| static double | [**invert**](http://docs.google.com/org/opencv/core/Core.html#invert(org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Finds the inverse or pseudo-inverse of a matrix. |
| static double | [**invert**](http://docs.google.com/org/opencv/core/Core.html#invert(org.opencv.core.Mat,%20org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int flags) Finds the inverse or pseudo-inverse of a matrix. |
| static double | [**kmeans**](http://docs.google.com/org/opencv/core/Core.html#kmeans(org.opencv.core.Mat,%20int,%20org.opencv.core.Mat,%20org.opencv.core.TermCriteria,%20int,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) data, int K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) bestLabels, [TermCriteria](http://docs.google.com/org/opencv/core/TermCriteria.html) criteria, int attempts, int flags) Finds centers of clusters and groups input samples around the clusters. |
| static double | [**kmeans**](http://docs.google.com/org/opencv/core/Core.html#kmeans(org.opencv.core.Mat,%20int,%20org.opencv.core.Mat,%20org.opencv.core.TermCriteria,%20int,%20int,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) data, int K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) bestLabels, [TermCriteria](http://docs.google.com/org/opencv/core/TermCriteria.html) criteria, int attempts, int flags, [Mat](http://docs.google.com/org/opencv/core/Mat.html) centers) Finds centers of clusters and groups input samples around the clusters. |
| static void | [**log**](http://docs.google.com/org/opencv/core/Core.html#log(org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Calculates the natural logarithm of every array element. |
| static void | [**LUT**](http://docs.google.com/org/opencv/core/Core.html#LUT(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) lut, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Performs a look-up table transform of an array. |
| static void | [**magnitude**](http://docs.google.com/org/opencv/core/Core.html#magnitude(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) x, [Mat](http://docs.google.com/org/opencv/core/Mat.html) y, [Mat](http://docs.google.com/org/opencv/core/Mat.html) magnitude) Calculates the magnitude of 2D vectors. |
| static double | [**Mahalanobis**](http://docs.google.com/org/opencv/core/Core.html#Mahalanobis(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) v1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) v2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) icovar) Calculates the Mahalanobis distance between two vectors. |
| static void | [**max**](http://docs.google.com/org/opencv/core/Core.html#max(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Calculates per-element maximum of two arrays or an array and a scalar. |
| static void | [**max**](http://docs.google.com/org/opencv/core/Core.html#max(org.opencv.core.Mat,%20org.opencv.core.Scalar,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) |
| static [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) | [**mean**](http://docs.google.com/org/opencv/core/Core.html#mean(org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src) Calculates an average (mean) of array elements. |
| static [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) | [**mean**](http://docs.google.com/org/opencv/core/Core.html#mean(org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) Calculates an average (mean) of array elements. |
| static void | [**meanStdDev**](http://docs.google.com/org/opencv/core/Core.html#meanStdDev(org.opencv.core.Mat,%20org.opencv.core.MatOfDouble,%20org.opencv.core.MatOfDouble))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [MatOfDouble](http://docs.google.com/org/opencv/core/MatOfDouble.html) mean, [MatOfDouble](http://docs.google.com/org/opencv/core/MatOfDouble.html) stddev) Calculates a mean and standard deviation of array elements. |
| static void | [**meanStdDev**](http://docs.google.com/org/opencv/core/Core.html#meanStdDev(org.opencv.core.Mat,%20org.opencv.core.MatOfDouble,%20org.opencv.core.MatOfDouble,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [MatOfDouble](http://docs.google.com/org/opencv/core/MatOfDouble.html) mean, [MatOfDouble](http://docs.google.com/org/opencv/core/MatOfDouble.html) stddev, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) Calculates a mean and standard deviation of array elements. |
| static void | [**merge**](http://docs.google.com/org/opencv/core/Core.html#merge(java.util.List,%20org.opencv.core.Mat))(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> mv, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) |
| static void | [**min**](http://docs.google.com/org/opencv/core/Core.html#min(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Calculates per-element minimum of two arrays or an array and a scalar. |
| static void | [**min**](http://docs.google.com/org/opencv/core/Core.html#min(org.opencv.core.Mat,%20org.opencv.core.Scalar,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) |
| static [Core.MinMaxLocResult](http://docs.google.com/org/opencv/core/Core.MinMaxLocResult.html) | [**minMaxLoc**](http://docs.google.com/org/opencv/core/Core.html#minMaxLoc(org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src) |
| static [Core.MinMaxLocResult](http://docs.google.com/org/opencv/core/Core.MinMaxLocResult.html) | [**minMaxLoc**](http://docs.google.com/org/opencv/core/Core.html#minMaxLoc(org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) |
| static void | [**mixChannels**](http://docs.google.com/org/opencv/core/Core.html#mixChannels(java.util.List,%20java.util.List,%20org.opencv.core.MatOfInt))(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> src, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> dst, [MatOfInt](http://docs.google.com/org/opencv/core/MatOfInt.html) fromTo) |
| static void | [**mulSpectrums**](http://docs.google.com/org/opencv/core/Core.html#mulSpectrums(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) a, [Mat](http://docs.google.com/org/opencv/core/Mat.html) b, [Mat](http://docs.google.com/org/opencv/core/Mat.html) c, int flags) Performs the per-element multiplication of two Fourier spectrums. |
| static void | [**mulSpectrums**](http://docs.google.com/org/opencv/core/Core.html#mulSpectrums(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20boolean))([Mat](http://docs.google.com/org/opencv/core/Mat.html) a, [Mat](http://docs.google.com/org/opencv/core/Mat.html) b, [Mat](http://docs.google.com/org/opencv/core/Mat.html) c, int flags, boolean conjB) Performs the per-element multiplication of two Fourier spectrums. |
| static void | [**multiply**](http://docs.google.com/org/opencv/core/Core.html#multiply(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Calculates the per-element scaled product of two arrays. |
| static void | [**multiply**](http://docs.google.com/org/opencv/core/Core.html#multiply(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double scale) Calculates the per-element scaled product of two arrays. |
| static void | [**multiply**](http://docs.google.com/org/opencv/core/Core.html#multiply(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20double,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double scale, int dtype) Calculates the per-element scaled product of two arrays. |
| static void | [**multiply**](http://docs.google.com/org/opencv/core/Core.html#multiply(org.opencv.core.Mat,%20org.opencv.core.Scalar,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) |
| static void | [**multiply**](http://docs.google.com/org/opencv/core/Core.html#multiply(org.opencv.core.Mat,%20org.opencv.core.Scalar,%20org.opencv.core.Mat,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double scale) |
| static void | [**multiply**](http://docs.google.com/org/opencv/core/Core.html#multiply(org.opencv.core.Mat,%20org.opencv.core.Scalar,%20org.opencv.core.Mat,%20double,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double scale, int dtype) |
| static void | [**mulTransposed**](http://docs.google.com/org/opencv/core/Core.html#mulTransposed(org.opencv.core.Mat,%20org.opencv.core.Mat,%20boolean))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, boolean aTa) Calculates the product of a matrix and its transposition. |
| static void | [**mulTransposed**](http://docs.google.com/org/opencv/core/Core.html#mulTransposed(org.opencv.core.Mat,%20org.opencv.core.Mat,%20boolean,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, boolean aTa, [Mat](http://docs.google.com/org/opencv/core/Mat.html) delta) Calculates the product of a matrix and its transposition. |
| static void | [**mulTransposed**](http://docs.google.com/org/opencv/core/Core.html#mulTransposed(org.opencv.core.Mat,%20org.opencv.core.Mat,%20boolean,%20org.opencv.core.Mat,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, boolean aTa, [Mat](http://docs.google.com/org/opencv/core/Mat.html) delta, double scale) Calculates the product of a matrix and its transposition. |
| static void | [**mulTransposed**](http://docs.google.com/org/opencv/core/Core.html#mulTransposed(org.opencv.core.Mat,%20org.opencv.core.Mat,%20boolean,%20org.opencv.core.Mat,%20double,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, boolean aTa, [Mat](http://docs.google.com/org/opencv/core/Mat.html) delta, double scale, int dtype) Calculates the product of a matrix and its transposition. |
| static double | [**norm**](http://docs.google.com/org/opencv/core/Core.html#norm(org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1) Calculates the absolute norm of an array. |
| static double | [**norm**](http://docs.google.com/org/opencv/core/Core.html#norm(org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, int normType) Calculates the absolute norm of an array. |
| static double | [**norm**](http://docs.google.com/org/opencv/core/Core.html#norm(org.opencv.core.Mat,%20int,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, int normType, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) Calculates the absolute norm of an array. |
| static double | [**norm**](http://docs.google.com/org/opencv/core/Core.html#norm(org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2) Calculates an absolute difference norm or a relative difference norm. |
| static double | [**norm**](http://docs.google.com/org/opencv/core/Core.html#norm(org.opencv.core.Mat,%20org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, int normType) Calculates an absolute difference norm or a relative difference norm. |
| static double | [**norm**](http://docs.google.com/org/opencv/core/Core.html#norm(org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, int normType, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) Calculates an absolute difference norm or a relative difference norm. |
| static void | [**normalize**](http://docs.google.com/org/opencv/core/Core.html#normalize(org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Normalizes the norm or value range of an array. |
| static void | [**normalize**](http://docs.google.com/org/opencv/core/Core.html#normalize(org.opencv.core.Mat,%20org.opencv.core.Mat,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double alpha) Normalizes the norm or value range of an array. |
| static void | [**normalize**](http://docs.google.com/org/opencv/core/Core.html#normalize(org.opencv.core.Mat,%20org.opencv.core.Mat,%20double,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double alpha, double beta) Normalizes the norm or value range of an array. |
| static void | [**normalize**](http://docs.google.com/org/opencv/core/Core.html#normalize(org.opencv.core.Mat,%20org.opencv.core.Mat,%20double,%20double,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double alpha, double beta, int norm\_type) Normalizes the norm or value range of an array. |
| static void | [**normalize**](http://docs.google.com/org/opencv/core/Core.html#normalize(org.opencv.core.Mat,%20org.opencv.core.Mat,%20double,%20double,%20int,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double alpha, double beta, int norm\_type, int dtype) Normalizes the norm or value range of an array. |
| static void | [**normalize**](http://docs.google.com/org/opencv/core/Core.html#normalize(org.opencv.core.Mat,%20org.opencv.core.Mat,%20double,%20double,%20int,%20int,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double alpha, double beta, int norm\_type, int dtype, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) Normalizes the norm or value range of an array. |
| static void | [**patchNaNs**](http://docs.google.com/org/opencv/core/Core.html#patchNaNs(org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) a) converts NaNs to the given number |
| static void | [**patchNaNs**](http://docs.google.com/org/opencv/core/Core.html#patchNaNs(org.opencv.core.Mat,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) a, double val) converts NaNs to the given number |
| static void | [**PCABackProject**](http://docs.google.com/org/opencv/core/Core.html#PCABackProject(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) data, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mean, [Mat](http://docs.google.com/org/opencv/core/Mat.html) eigenvectors, [Mat](http://docs.google.com/org/opencv/core/Mat.html) result) wrap PCA::backProject |
| static void | [**PCACompute**](http://docs.google.com/org/opencv/core/Core.html#PCACompute(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) data, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mean, [Mat](http://docs.google.com/org/opencv/core/Mat.html) eigenvectors) wrap PCA::operator() |
| static void | [**PCACompute**](http://docs.google.com/org/opencv/core/Core.html#PCACompute(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) data, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mean, [Mat](http://docs.google.com/org/opencv/core/Mat.html) eigenvectors, double retainedVariance) wrap PCA::operator() |
| static void | [**PCACompute**](http://docs.google.com/org/opencv/core/Core.html#PCACompute(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) data, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mean, [Mat](http://docs.google.com/org/opencv/core/Mat.html) eigenvectors, int maxComponents) wrap PCA::operator() |
| static void | [**PCACompute2**](http://docs.google.com/org/opencv/core/Core.html#PCACompute2(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) data, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mean, [Mat](http://docs.google.com/org/opencv/core/Mat.html) eigenvectors, [Mat](http://docs.google.com/org/opencv/core/Mat.html) eigenvalues) wrap PCA::operator() and add eigenvalues output parameter |
| static void | [**PCACompute2**](http://docs.google.com/org/opencv/core/Core.html#PCACompute2(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) data, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mean, [Mat](http://docs.google.com/org/opencv/core/Mat.html) eigenvectors, [Mat](http://docs.google.com/org/opencv/core/Mat.html) eigenvalues, double retainedVariance) wrap PCA::operator() and add eigenvalues output parameter |
| static void | [**PCACompute2**](http://docs.google.com/org/opencv/core/Core.html#PCACompute2(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) data, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mean, [Mat](http://docs.google.com/org/opencv/core/Mat.html) eigenvectors, [Mat](http://docs.google.com/org/opencv/core/Mat.html) eigenvalues, int maxComponents) wrap PCA::operator() and add eigenvalues output parameter |
| static void | [**PCAProject**](http://docs.google.com/org/opencv/core/Core.html#PCAProject(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) data, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mean, [Mat](http://docs.google.com/org/opencv/core/Mat.html) eigenvectors, [Mat](http://docs.google.com/org/opencv/core/Mat.html) result) wrap PCA::project |
| static void | [**perspectiveTransform**](http://docs.google.com/org/opencv/core/Core.html#perspectiveTransform(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) m) Performs the perspective matrix transformation of vectors. |
| static void | [**phase**](http://docs.google.com/org/opencv/core/Core.html#phase(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) x, [Mat](http://docs.google.com/org/opencv/core/Mat.html) y, [Mat](http://docs.google.com/org/opencv/core/Mat.html) angle) Calculates the rotation angle of 2D vectors. |
| static void | [**phase**](http://docs.google.com/org/opencv/core/Core.html#phase(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20boolean))([Mat](http://docs.google.com/org/opencv/core/Mat.html) x, [Mat](http://docs.google.com/org/opencv/core/Mat.html) y, [Mat](http://docs.google.com/org/opencv/core/Mat.html) angle, boolean angleInDegrees) Calculates the rotation angle of 2D vectors. |
| static void | [**polarToCart**](http://docs.google.com/org/opencv/core/Core.html#polarToCart(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) magnitude, [Mat](http://docs.google.com/org/opencv/core/Mat.html) angle, [Mat](http://docs.google.com/org/opencv/core/Mat.html) x, [Mat](http://docs.google.com/org/opencv/core/Mat.html) y) Calculates x and y coordinates of 2D vectors from their magnitude and angle. |
| static void | [**polarToCart**](http://docs.google.com/org/opencv/core/Core.html#polarToCart(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20boolean))([Mat](http://docs.google.com/org/opencv/core/Mat.html) magnitude, [Mat](http://docs.google.com/org/opencv/core/Mat.html) angle, [Mat](http://docs.google.com/org/opencv/core/Mat.html) x, [Mat](http://docs.google.com/org/opencv/core/Mat.html) y, boolean angleInDegrees) Calculates x and y coordinates of 2D vectors from their magnitude and angle. |
| static void | [**pow**](http://docs.google.com/org/opencv/core/Core.html#pow(org.opencv.core.Mat,%20double,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, double power, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Raises every array element to a power. |
| static double | [**PSNR**](http://docs.google.com/org/opencv/core/Core.html#PSNR(org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2) Computes the Peak Signal-to-Noise Ratio (PSNR) image quality metric. |
| static void | [**randn**](http://docs.google.com/org/opencv/core/Core.html#randn(org.opencv.core.Mat,%20double,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double mean, double stddev) Fills the array with normally distributed random numbers. |
| static void | [**randShuffle**](http://docs.google.com/org/opencv/core/Core.html#randShuffle(org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Shuffles the array elements randomly. |
| static void | [**randShuffle**](http://docs.google.com/org/opencv/core/Core.html#randShuffle(org.opencv.core.Mat,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double iterFactor) Shuffles the array elements randomly. |
| static void | [**randu**](http://docs.google.com/org/opencv/core/Core.html#randu(org.opencv.core.Mat,%20double,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double low, double high) Generates a single uniformly-distributed random number or an array of random numbers. |
| static void | [**reduce**](http://docs.google.com/org/opencv/core/Core.html#reduce(org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int dim, int rtype) Reduces a matrix to a vector. |
| static void | [**reduce**](http://docs.google.com/org/opencv/core/Core.html#reduce(org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20int,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int dim, int rtype, int dtype) Reduces a matrix to a vector. |
| static void | [**repeat**](http://docs.google.com/org/opencv/core/Core.html#repeat(org.opencv.core.Mat,%20int,%20int,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, int ny, int nx, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Fills the output array with repeated copies of the input array. |
| static void | [**rotate**](http://docs.google.com/org/opencv/core/Core.html#rotate(org.opencv.core.Mat,%20org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int rotateCode) Rotates a 2D array in multiples of 90 degrees. |
| static void | [**scaleAdd**](http://docs.google.com/org/opencv/core/Core.html#scaleAdd(org.opencv.core.Mat,%20double,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, double alpha, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Calculates the sum of a scaled array and another array. |
| static void | [**setErrorVerbosity**](http://docs.google.com/org/opencv/core/Core.html#setErrorVerbosity(boolean))(boolean verbose) |
| static void | [**setIdentity**](http://docs.google.com/org/opencv/core/Core.html#setIdentity(org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) mtx) Initializes a scaled identity matrix. |
| static void | [**setIdentity**](http://docs.google.com/org/opencv/core/Core.html#setIdentity(org.opencv.core.Mat,%20org.opencv.core.Scalar))([Mat](http://docs.google.com/org/opencv/core/Mat.html) mtx, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) s) Initializes a scaled identity matrix. |
| static void | [**setNumThreads**](http://docs.google.com/org/opencv/core/Core.html#setNumThreads(int))(int nthreads) OpenCV will try to set the number of threads for the next parallel region. |
| static void | [**setRNGSeed**](http://docs.google.com/org/opencv/core/Core.html#setRNGSeed(int))(int seed) Sets state of default random number generator. |
| static void | [**setUseIPP\_NE**](http://docs.google.com/org/opencv/core/Core.html#setUseIPP_NE(boolean))(boolean flag) |
| static void | [**setUseIPP\_NotExact**](http://docs.google.com/org/opencv/core/Core.html#setUseIPP_NotExact(boolean))(boolean flag) |
| static void | [**setUseIPP**](http://docs.google.com/org/opencv/core/Core.html#setUseIPP(boolean))(boolean flag) |
| static boolean | [**solve**](http://docs.google.com/org/opencv/core/Core.html#solve(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Solves one or more linear systems or least-squares problems. |
| static boolean | [**solve**](http://docs.google.com/org/opencv/core/Core.html#solve(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int flags) Solves one or more linear systems or least-squares problems. |
| static int | [**solveCubic**](http://docs.google.com/org/opencv/core/Core.html#solveCubic(org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) coeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) roots) Finds the real roots of a cubic equation. |
| static double | [**solvePoly**](http://docs.google.com/org/opencv/core/Core.html#solvePoly(org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) coeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) roots) Finds the real or complex roots of a polynomial equation. |
| static double | [**solvePoly**](http://docs.google.com/org/opencv/core/Core.html#solvePoly(org.opencv.core.Mat,%20org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) coeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) roots, int maxIters) Finds the real or complex roots of a polynomial equation. |
| static void | [**sort**](http://docs.google.com/org/opencv/core/Core.html#sort(org.opencv.core.Mat,%20org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int flags) Sorts each row or each column of a matrix. |
| static void | [**sortIdx**](http://docs.google.com/org/opencv/core/Core.html#sortIdx(org.opencv.core.Mat,%20org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int flags) Sorts each row or each column of a matrix. |
| static void | [**split**](http://docs.google.com/org/opencv/core/Core.html#split(org.opencv.core.Mat,%20java.util.List))([Mat](http://docs.google.com/org/opencv/core/Mat.html) m, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> mv) |
| static void | [**sqrt**](http://docs.google.com/org/opencv/core/Core.html#sqrt(org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Calculates a square root of array elements. |
| static void | [**subtract**](http://docs.google.com/org/opencv/core/Core.html#subtract(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Calculates the per-element difference between two arrays or array and a scalar. |
| static void | [**subtract**](http://docs.google.com/org/opencv/core/Core.html#subtract(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) Calculates the per-element difference between two arrays or array and a scalar. |
| static void | [**subtract**](http://docs.google.com/org/opencv/core/Core.html#subtract(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask, int dtype) Calculates the per-element difference between two arrays or array and a scalar. |
| static void | [**subtract**](http://docs.google.com/org/opencv/core/Core.html#subtract(org.opencv.core.Mat,%20org.opencv.core.Scalar,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) |
| static void | [**subtract**](http://docs.google.com/org/opencv/core/Core.html#subtract(org.opencv.core.Mat,%20org.opencv.core.Scalar,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) |
| static void | [**subtract**](http://docs.google.com/org/opencv/core/Core.html#subtract(org.opencv.core.Mat,%20org.opencv.core.Scalar,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask, int dtype) |
| static [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) | [**sumElems**](http://docs.google.com/org/opencv/core/Core.html#sumElems(org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src) Calculates the sum of array elements. |
| static void | [**SVBackSubst**](http://docs.google.com/org/opencv/core/Core.html#SVBackSubst(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) w, [Mat](http://docs.google.com/org/opencv/core/Mat.html) u, [Mat](http://docs.google.com/org/opencv/core/Mat.html) vt, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rhs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) wrap SVD::backSubst |
| static void | [**SVDecomp**](http://docs.google.com/org/opencv/core/Core.html#SVDecomp(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) w, [Mat](http://docs.google.com/org/opencv/core/Mat.html) u, [Mat](http://docs.google.com/org/opencv/core/Mat.html) vt) wrap SVD::compute |
| static void | [**SVDecomp**](http://docs.google.com/org/opencv/core/Core.html#SVDecomp(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) w, [Mat](http://docs.google.com/org/opencv/core/Mat.html) u, [Mat](http://docs.google.com/org/opencv/core/Mat.html) vt, int flags) wrap SVD::compute |
| static [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) | [**trace**](http://docs.google.com/org/opencv/core/Core.html#trace(org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) mtx) Returns the trace of a matrix. |
| static void | [**transform**](http://docs.google.com/org/opencv/core/Core.html#transform(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) m) Performs the matrix transformation of every array element. |
| static void | [**transpose**](http://docs.google.com/org/opencv/core/Core.html#transpose(org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Transposes a matrix. |
| static boolean | [**useIPP\_NE**](http://docs.google.com/org/opencv/core/Core.html#useIPP_NE())() |
| static boolean | [**useIPP\_NotExact**](http://docs.google.com/org/opencv/core/Core.html#useIPP_NotExact())() |
| static boolean | [**useIPP**](http://docs.google.com/org/opencv/core/Core.html#useIPP())() proxy for hal::Cholesky |
| static void | [**vconcat**](http://docs.google.com/org/opencv/core/Core.html#vconcat(java.util.List,%20org.opencv.core.Mat))(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) std::vector<cv::Mat> matrices = { cv::Mat(1, 4, CV\_8UC1, cv::Scalar(1)), cv::Mat(1, 4, CV\_8UC1, cv::Scalar(2)), cv::Mat(1, 4, CV\_8UC1, cv::Scalar(3)),}; cv::Mat out; cv::vconcat( matrices, out ); //out: //[1, 1, 1, 1; // 2, 2, 2, 2; // 3, 3, 3, 3] |

### Methods inherited from class java.lang.Objectequals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

### Field Detail

#### BadAlign public static final int BadAlignSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.BadAlign)

#### BadAlphaChannel public static final int BadAlphaChannelSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.BadAlphaChannel)

#### BadCallBack public static final int BadCallBackSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.BadCallBack)

#### BadCOI public static final int BadCOISee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.BadCOI)

#### BadDataPtr public static final int BadDataPtrSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.BadDataPtr)

#### BadDepth public static final int BadDepthSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.BadDepth)

#### BadImageSize public static final int BadImageSizeSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.BadImageSize)

#### BadModelOrChSeq public static final int BadModelOrChSeqSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.BadModelOrChSeq)

#### BadNumChannel1U public static final int BadNumChannel1USee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.BadNumChannel1U)

#### BadNumChannels public static final int BadNumChannelsSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.BadNumChannels)

#### BadOffset public static final int BadOffsetSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.BadOffset)

#### BadOrder public static final int BadOrderSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.BadOrder)

#### BadOrigin public static final int BadOriginSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.BadOrigin)

#### BadROISize public static final int BadROISizeSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.BadROISize)

#### BadStep public static final int BadStepSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.BadStep)

#### BadTileSize public static final int BadTileSizeSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.BadTileSize)

#### BORDER\_CONSTANT public static final int BORDER\_CONSTANTSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.BORDER_CONSTANT)

#### BORDER\_DEFAULT public static final int BORDER\_DEFAULTSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.BORDER_DEFAULT)

#### BORDER\_ISOLATED public static final int BORDER\_ISOLATEDSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.BORDER_ISOLATED)

#### BORDER\_REFLECT public static final int BORDER\_REFLECTSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.BORDER_REFLECT)

#### BORDER\_REFLECT\_101 public static final int BORDER\_REFLECT\_101See Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.BORDER_REFLECT_101)

#### BORDER\_REFLECT101 public static final int BORDER\_REFLECT101See Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.BORDER_REFLECT101)

#### BORDER\_REPLICATE public static final int BORDER\_REPLICATESee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.BORDER_REPLICATE)

#### BORDER\_TRANSPARENT public static final int BORDER\_TRANSPARENTSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.BORDER_TRANSPARENT)

#### BORDER\_WRAP public static final int BORDER\_WRAPSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.BORDER_WRAP)

#### CMP\_EQ public static final int CMP\_EQSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.CMP_EQ)

#### CMP\_GE public static final int CMP\_GESee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.CMP_GE)

#### CMP\_GT public static final int CMP\_GTSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.CMP_GT)

#### CMP\_LE public static final int CMP\_LESee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.CMP_LE)

#### CMP\_LT public static final int CMP\_LTSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.CMP_LT)

#### CMP\_NE public static final int CMP\_NESee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.CMP_NE)

#### COVAR\_COLS public static final int COVAR\_COLSSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.COVAR_COLS)

#### COVAR\_NORMAL public static final int COVAR\_NORMALSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.COVAR_NORMAL)

#### COVAR\_ROWS public static final int COVAR\_ROWSSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.COVAR_ROWS)

#### COVAR\_SCALE public static final int COVAR\_SCALESee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.COVAR_SCALE)

#### COVAR\_SCRAMBLED public static final int COVAR\_SCRAMBLEDSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.COVAR_SCRAMBLED)

#### COVAR\_USE\_AVG public static final int COVAR\_USE\_AVGSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.COVAR_USE_AVG)

#### DCT\_INVERSE public static final int DCT\_INVERSESee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.DCT_INVERSE)

#### DCT\_ROWS public static final int DCT\_ROWSSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.DCT_ROWS)

#### DECOMP\_CHOLESKY public static final int DECOMP\_CHOLESKYSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.DECOMP_CHOLESKY)

#### DECOMP\_EIG public static final int DECOMP\_EIGSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.DECOMP_EIG)

#### DECOMP\_LU public static final int DECOMP\_LUSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.DECOMP_LU)

#### DECOMP\_NORMAL public static final int DECOMP\_NORMALSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.DECOMP_NORMAL)

#### DECOMP\_QR public static final int DECOMP\_QRSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.DECOMP_QR)

#### DECOMP\_SVD public static final int DECOMP\_SVDSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.DECOMP_SVD)

#### DFT\_COMPLEX\_INPUT public static final int DFT\_COMPLEX\_INPUTSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.DFT_COMPLEX_INPUT)

#### DFT\_COMPLEX\_OUTPUT public static final int DFT\_COMPLEX\_OUTPUTSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.DFT_COMPLEX_OUTPUT)

#### DFT\_INVERSE public static final int DFT\_INVERSESee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.DFT_INVERSE)

#### DFT\_REAL\_OUTPUT public static final int DFT\_REAL\_OUTPUTSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.DFT_REAL_OUTPUT)

#### DFT\_ROWS public static final int DFT\_ROWSSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.DFT_ROWS)

#### DFT\_SCALE public static final int DFT\_SCALESee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.DFT_SCALE)

#### FILLED public static final int FILLEDSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.FILLED)

#### FONT\_HERSHEY\_COMPLEX public static final int FONT\_HERSHEY\_COMPLEXSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.FONT_HERSHEY_COMPLEX)

#### FONT\_HERSHEY\_COMPLEX\_SMALL public static final int FONT\_HERSHEY\_COMPLEX\_SMALLSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.FONT_HERSHEY_COMPLEX_SMALL)

#### FONT\_HERSHEY\_DUPLEX public static final int FONT\_HERSHEY\_DUPLEXSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.FONT_HERSHEY_DUPLEX)

#### FONT\_HERSHEY\_PLAIN public static final int FONT\_HERSHEY\_PLAINSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.FONT_HERSHEY_PLAIN)

#### FONT\_HERSHEY\_SCRIPT\_COMPLEX public static final int FONT\_HERSHEY\_SCRIPT\_COMPLEXSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.FONT_HERSHEY_SCRIPT_COMPLEX)

#### FONT\_HERSHEY\_SCRIPT\_SIMPLEX public static final int FONT\_HERSHEY\_SCRIPT\_SIMPLEXSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.FONT_HERSHEY_SCRIPT_SIMPLEX)

#### FONT\_HERSHEY\_SIMPLEX public static final int FONT\_HERSHEY\_SIMPLEXSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.FONT_HERSHEY_SIMPLEX)

#### FONT\_HERSHEY\_TRIPLEX public static final int FONT\_HERSHEY\_TRIPLEXSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.FONT_HERSHEY_TRIPLEX)

#### FONT\_ITALIC public static final int FONT\_ITALICSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.FONT_ITALIC)

#### Formatter\_FMT\_C public static final int Formatter\_FMT\_CSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.Formatter_FMT_C)

#### Formatter\_FMT\_CSV public static final int Formatter\_FMT\_CSVSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.Formatter_FMT_CSV)

#### Formatter\_FMT\_DEFAULT public static final int Formatter\_FMT\_DEFAULTSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.Formatter_FMT_DEFAULT)

#### Formatter\_FMT\_MATLAB public static final int Formatter\_FMT\_MATLABSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.Formatter_FMT_MATLAB)

#### Formatter\_FMT\_NUMPY public static final int Formatter\_FMT\_NUMPYSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.Formatter_FMT_NUMPY)

#### Formatter\_FMT\_PYTHON public static final int Formatter\_FMT\_PYTHONSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.Formatter_FMT_PYTHON)

#### GEMM\_1\_T public static final int GEMM\_1\_TSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.GEMM_1_T)

#### GEMM\_2\_T public static final int GEMM\_2\_TSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.GEMM_2_T)

#### GEMM\_3\_T public static final int GEMM\_3\_TSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.GEMM_3_T)

#### GpuApiCallError public static final int GpuApiCallErrorSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.GpuApiCallError)

#### GpuNotSupported public static final int GpuNotSupportedSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.GpuNotSupported)

#### Hamming\_normType public static final int Hamming\_normTypeSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.Hamming_normType)

#### HeaderIsNull public static final int HeaderIsNullSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.HeaderIsNull)

#### KMEANS\_PP\_CENTERS public static final int KMEANS\_PP\_CENTERSSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.KMEANS_PP_CENTERS)

#### KMEANS\_RANDOM\_CENTERS public static final int KMEANS\_RANDOM\_CENTERSSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.KMEANS_RANDOM_CENTERS)

#### KMEANS\_USE\_INITIAL\_LABELS public static final int KMEANS\_USE\_INITIAL\_LABELSSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.KMEANS_USE_INITIAL_LABELS)

#### LINE\_4 public static final int LINE\_4See Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.LINE_4)

#### LINE\_8 public static final int LINE\_8See Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.LINE_8)

#### LINE\_AA public static final int LINE\_AASee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.LINE_AA)

#### MaskIsTiled public static final int MaskIsTiledSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.MaskIsTiled)

#### NATIVE\_LIBRARY\_NAME public static final java.lang.String NATIVE\_LIBRARY\_NAME

#### NORM\_HAMMING public static final int NORM\_HAMMINGSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.NORM_HAMMING)

#### NORM\_HAMMING2 public static final int NORM\_HAMMING2See Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.NORM_HAMMING2)

#### NORM\_INF public static final int NORM\_INFSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.NORM_INF)

#### NORM\_L1 public static final int NORM\_L1See Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.NORM_L1)

#### NORM\_L2 public static final int NORM\_L2See Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.NORM_L2)

#### NORM\_L2SQR public static final int NORM\_L2SQRSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.NORM_L2SQR)

#### NORM\_MINMAX public static final int NORM\_MINMAXSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.NORM_MINMAX)

#### NORM\_RELATIVE public static final int NORM\_RELATIVESee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.NORM_RELATIVE)

#### NORM\_TYPE\_MASK public static final int NORM\_TYPE\_MASKSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.NORM_TYPE_MASK)

#### OpenCLApiCallError public static final int OpenCLApiCallErrorSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.OpenCLApiCallError)

#### OpenCLDoubleNotSupported public static final int OpenCLDoubleNotSupportedSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.OpenCLDoubleNotSupported)

#### OpenCLInitError public static final int OpenCLInitErrorSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.OpenCLInitError)

#### OpenCLNoAMDBlasFft public static final int OpenCLNoAMDBlasFftSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.OpenCLNoAMDBlasFft)

#### OpenGlApiCallError public static final int OpenGlApiCallErrorSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.OpenGlApiCallError)

#### OpenGlNotSupported public static final int OpenGlNotSupportedSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.OpenGlNotSupported)

#### Param\_ALGORITHM public static final int Param\_ALGORITHMSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.Param_ALGORITHM)

#### Param\_BOOLEAN public static final int Param\_BOOLEANSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.Param_BOOLEAN)

#### Param\_FLOAT public static final int Param\_FLOATSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.Param_FLOAT)

#### Param\_INT public static final int Param\_INTSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.Param_INT)

#### Param\_MAT public static final int Param\_MATSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.Param_MAT)

#### Param\_MAT\_VECTOR public static final int Param\_MAT\_VECTORSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.Param_MAT_VECTOR)

#### Param\_REAL public static final int Param\_REALSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.Param_REAL)

#### Param\_SCALAR public static final int Param\_SCALARSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.Param_SCALAR)

#### Param\_STRING public static final int Param\_STRINGSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.Param_STRING)

#### Param\_UCHAR public static final int Param\_UCHARSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.Param_UCHAR)

#### Param\_UINT64 public static final int Param\_UINT64See Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.Param_UINT64)

#### Param\_UNSIGNED\_INT public static final int Param\_UNSIGNED\_INTSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.Param_UNSIGNED_INT)

#### PCA\_DATA\_AS\_COL public static final int PCA\_DATA\_AS\_COLSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.PCA_DATA_AS_COL)

#### PCA\_DATA\_AS\_ROW public static final int PCA\_DATA\_AS\_ROWSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.PCA_DATA_AS_ROW)

#### PCA\_USE\_AVG public static final int PCA\_USE\_AVGSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.PCA_USE_AVG)

#### REDUCE\_AVG public static final int REDUCE\_AVGSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.REDUCE_AVG)

#### REDUCE\_MAX public static final int REDUCE\_MAXSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.REDUCE_MAX)

#### REDUCE\_MIN public static final int REDUCE\_MINSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.REDUCE_MIN)

#### REDUCE\_SUM public static final int REDUCE\_SUMSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.REDUCE_SUM)

#### RNG\_NORMAL public static final int RNG\_NORMALSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.RNG_NORMAL)

#### RNG\_UNIFORM public static final int RNG\_UNIFORMSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.RNG_UNIFORM)

#### ROTATE\_180 public static final int ROTATE\_180See Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.ROTATE_180)

#### ROTATE\_90\_CLOCKWISE public static final int ROTATE\_90\_CLOCKWISESee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.ROTATE_90_CLOCKWISE)

#### ROTATE\_90\_COUNTERCLOCKWISE public static final int ROTATE\_90\_COUNTERCLOCKWISESee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.ROTATE_90_COUNTERCLOCKWISE)

#### SORT\_ASCENDING public static final int SORT\_ASCENDINGSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.SORT_ASCENDING)

#### SORT\_DESCENDING public static final int SORT\_DESCENDINGSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.SORT_DESCENDING)

#### SORT\_EVERY\_COLUMN public static final int SORT\_EVERY\_COLUMNSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.SORT_EVERY_COLUMN)

#### SORT\_EVERY\_ROW public static final int SORT\_EVERY\_ROWSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.SORT_EVERY_ROW)

#### StsAssert public static final int StsAssertSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.StsAssert)

#### StsAutoTrace public static final int StsAutoTraceSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.StsAutoTrace)

#### StsBackTrace public static final int StsBackTraceSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.StsBackTrace)

#### StsBadArg public static final int StsBadArgSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.StsBadArg)

#### StsBadFlag public static final int StsBadFlagSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.StsBadFlag)

#### StsBadFunc public static final int StsBadFuncSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.StsBadFunc)

#### StsBadMask public static final int StsBadMaskSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.StsBadMask)

#### StsBadMemBlock public static final int StsBadMemBlockSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.StsBadMemBlock)

#### StsBadPoint public static final int StsBadPointSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.StsBadPoint)

#### StsBadSize public static final int StsBadSizeSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.StsBadSize)

#### StsDivByZero public static final int StsDivByZeroSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.StsDivByZero)

#### StsError public static final int StsErrorSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.StsError)

#### StsFilterOffsetErr public static final int StsFilterOffsetErrSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.StsFilterOffsetErr)

#### StsFilterStructContentErr public static final int StsFilterStructContentErrSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.StsFilterStructContentErr)

#### StsInplaceNotSupported public static final int StsInplaceNotSupportedSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.StsInplaceNotSupported)

#### StsInternal public static final int StsInternalSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.StsInternal)

#### StsKernelStructContentErr public static final int StsKernelStructContentErrSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.StsKernelStructContentErr)

#### StsNoConv public static final int StsNoConvSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.StsNoConv)

#### StsNoMem public static final int StsNoMemSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.StsNoMem)

#### StsNotImplemented public static final int StsNotImplementedSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.StsNotImplemented)

#### StsNullPtr public static final int StsNullPtrSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.StsNullPtr)

#### StsObjectNotFound public static final int StsObjectNotFoundSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.StsObjectNotFound)

#### StsOk public static final int StsOkSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.StsOk)

#### StsOutOfRange public static final int StsOutOfRangeSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.StsOutOfRange)

#### StsParseError public static final int StsParseErrorSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.StsParseError)

#### StsUnmatchedFormats public static final int StsUnmatchedFormatsSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.StsUnmatchedFormats)

#### StsUnmatchedSizes public static final int StsUnmatchedSizesSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.StsUnmatchedSizes)

#### StsUnsupportedFormat public static final int StsUnsupportedFormatSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.StsUnsupportedFormat)

#### StsVecLengthErr public static final int StsVecLengthErrSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.StsVecLengthErr)

#### SVD\_FULL\_UV public static final int SVD\_FULL\_UVSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.SVD_FULL_UV)

#### SVD\_MODIFY\_A public static final int SVD\_MODIFY\_ASee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.SVD_MODIFY_A)

#### SVD\_NO\_UV public static final int SVD\_NO\_UVSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.core.Core.SVD_NO_UV)

#### VERSION public static final java.lang.String VERSION

#### VERSION\_MAJOR public static final int VERSION\_MAJOR

#### VERSION\_MINOR public static final int VERSION\_MINOR

#### VERSION\_REVISION public static final int VERSION\_REVISION

#### VERSION\_STATUS public static final java.lang.String VERSION\_STATUS

### Constructor Detail

#### Core public Core()

### Method Detail

#### absdiff public static void absdiff([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Calculates the per-element absolute difference between two arrays or between an array and a scalar. The function cv::absdiff calculates: Absolute difference between two arrays when they have the same size and type: \(\texttt{dst}(I) = \texttt{saturate} (| \texttt{src1}(I) - \texttt{src2}(I)|)\) Absolute difference between an array and a scalar when the second array is constructed from Scalar or has as many elements as the number of channels in src1: \(\texttt{dst}(I) = \texttt{saturate} (| \texttt{src1}(I) - \texttt{src2} |)\) Absolute difference between a scalar and an array when the first array is constructed from Scalar or has as many elements as the number of channels in src2: \(\texttt{dst}(I) = \texttt{saturate} (| \texttt{src1} - \texttt{src2}(I) |)\) where I is a multi-dimensional index of array elements. In case of multi-channel arrays, each channel is processed independently. **Note:** Saturation is not applied when the arrays have the depth CV\_32S. You may even get a negative value in the case of overflow.Parameters:src1 - first input array or a scalar.src2 - second input array or a scalar.dst - output array that has the same size and type as input arrays. SEE: cv::abs(const Mat&)

#### absdiff public static void absdiff([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst)

#### add public static void add([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst)

Calculates the per-element sum of two arrays or an array and a scalar. The function add calculates:

* + - * Sum of two arrays when both input arrays have the same size and the same number of channels: \(\texttt{dst}(I) = \texttt{saturate} ( \texttt{src1}(I) + \texttt{src2}(I)) \quad \texttt{if mask}(I) \ne0\)
      * Sum of an array and a scalar when src2 is constructed from Scalar or has the same number of elements as src1.channels(): \(\texttt{dst}(I) = \texttt{saturate} ( \texttt{src1}(I) + \texttt{src2} ) \quad \texttt{if mask}(I) \ne0\)
      * Sum of a scalar and an array when src1 is constructed from Scalar or has the same number of elements as src2.channels(): \(\texttt{dst}(I) = \texttt{saturate} ( \texttt{src1} + \texttt{src2}(I) ) \quad \texttt{if mask}(I) \ne0\) where I is a multi-dimensional index of array elements. In case of multi-channel arrays, each channel is processed independently.

The first function in the list above can be replaced with matrix expressions: dst = src1 + src2; dst += src1; // equivalent to add(dst, src1, dst); The input arrays and the output array can all have the same or different depths. For example, you can add a 16-bit unsigned array to a 8-bit signed array and store the sum as a 32-bit floating-point array. Depth of the output array is determined by the dtype parameter. In the second and third cases above, as well as in the first case, when src1.depth() == src2.depth(), dtype can be set to the default -1. In this case, the output array will have the same depth as the input array, be it src1, src2 or both. **Note:** Saturation is not applied when the output array has the depth CV\_32S. You may even get result of an incorrect sign in the case of overflow.Parameters:src1 - first input array or a scalar.src2 - second input array or a scalar.dst - output array that has the same size and number of channels as the input array(s); the depth is defined by dtype or src1/src2. output array to be changed. SEE: subtract, addWeighted, scaleAdd, Mat::convertTo

#### add public static void add([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask)

Calculates the per-element sum of two arrays or an array and a scalar. The function add calculates:

* + - * Sum of two arrays when both input arrays have the same size and the same number of channels: \(\texttt{dst}(I) = \texttt{saturate} ( \texttt{src1}(I) + \texttt{src2}(I)) \quad \texttt{if mask}(I) \ne0\)
      * Sum of an array and a scalar when src2 is constructed from Scalar or has the same number of elements as src1.channels(): \(\texttt{dst}(I) = \texttt{saturate} ( \texttt{src1}(I) + \texttt{src2} ) \quad \texttt{if mask}(I) \ne0\)
      * Sum of a scalar and an array when src1 is constructed from Scalar or has the same number of elements as src2.channels(): \(\texttt{dst}(I) = \texttt{saturate} ( \texttt{src1} + \texttt{src2}(I) ) \quad \texttt{if mask}(I) \ne0\) where I is a multi-dimensional index of array elements. In case of multi-channel arrays, each channel is processed independently.

The first function in the list above can be replaced with matrix expressions: dst = src1 + src2; dst += src1; // equivalent to add(dst, src1, dst); The input arrays and the output array can all have the same or different depths. For example, you can add a 16-bit unsigned array to a 8-bit signed array and store the sum as a 32-bit floating-point array. Depth of the output array is determined by the dtype parameter. In the second and third cases above, as well as in the first case, when src1.depth() == src2.depth(), dtype can be set to the default -1. In this case, the output array will have the same depth as the input array, be it src1, src2 or both. **Note:** Saturation is not applied when the output array has the depth CV\_32S. You may even get result of an incorrect sign in the case of overflow.Parameters:src1 - first input array or a scalar.src2 - second input array or a scalar.dst - output array that has the same size and number of channels as the input array(s); the depth is defined by dtype or src1/src2.mask - optional operation mask - 8-bit single channel array, that specifies elements of the output array to be changed. SEE: subtract, addWeighted, scaleAdd, Mat::convertTo

#### add public static void add([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask, int dtype)

Calculates the per-element sum of two arrays or an array and a scalar. The function add calculates:

* + - * Sum of two arrays when both input arrays have the same size and the same number of channels: \(\texttt{dst}(I) = \texttt{saturate} ( \texttt{src1}(I) + \texttt{src2}(I)) \quad \texttt{if mask}(I) \ne0\)
      * Sum of an array and a scalar when src2 is constructed from Scalar or has the same number of elements as src1.channels(): \(\texttt{dst}(I) = \texttt{saturate} ( \texttt{src1}(I) + \texttt{src2} ) \quad \texttt{if mask}(I) \ne0\)
      * Sum of a scalar and an array when src1 is constructed from Scalar or has the same number of elements as src2.channels(): \(\texttt{dst}(I) = \texttt{saturate} ( \texttt{src1} + \texttt{src2}(I) ) \quad \texttt{if mask}(I) \ne0\) where I is a multi-dimensional index of array elements. In case of multi-channel arrays, each channel is processed independently.

The first function in the list above can be replaced with matrix expressions: dst = src1 + src2; dst += src1; // equivalent to add(dst, src1, dst); The input arrays and the output array can all have the same or different depths. For example, you can add a 16-bit unsigned array to a 8-bit signed array and store the sum as a 32-bit floating-point array. Depth of the output array is determined by the dtype parameter. In the second and third cases above, as well as in the first case, when src1.depth() == src2.depth(), dtype can be set to the default -1. In this case, the output array will have the same depth as the input array, be it src1, src2 or both. **Note:** Saturation is not applied when the output array has the depth CV\_32S. You may even get result of an incorrect sign in the case of overflow.Parameters:src1 - first input array or a scalar.src2 - second input array or a scalar.dst - output array that has the same size and number of channels as the input array(s); the depth is defined by dtype or src1/src2.mask - optional operation mask - 8-bit single channel array, that specifies elements of the output array to be changed.dtype - optional depth of the output array (see the discussion below). SEE: subtract, addWeighted, scaleAdd, Mat::convertTo

#### add public static void add([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst)

#### add public static void add([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask)

#### add public static void add([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask, int dtype)

#### addSamplesDataSearchPath public static void addSamplesDataSearchPath(java.lang.String path) Override search data path by adding new search location Use this only to override default behavior Passed paths are used in LIFO order.Parameters:path - Path to used samples data

#### addSamplesDataSearchSubDirectory public static void addSamplesDataSearchSubDirectory(java.lang.String subdir) Append samples search data sub directory General usage is to add OpenCV modules name (<opencv\_contrib>/modules/<name>/samples/data -> <name>/samples/data + modules/<name>/samples/data). Passed subdirectories are used in LIFO order.Parameters:subdir - samples data sub directory

#### addWeighted public static void addWeighted([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, double alpha, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, double beta, double gamma, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Calculates the weighted sum of two arrays. The function addWeighted calculates the weighted sum of two arrays as follows: \(\texttt{dst} (I)= \texttt{saturate} ( \texttt{src1} (I)\* \texttt{alpha} + \texttt{src2} (I)\* \texttt{beta} + \texttt{gamma} )\) where I is a multi-dimensional index of array elements. In case of multi-channel arrays, each channel is processed independently. The function can be replaced with a matrix expression: dst = src1\*alpha + src2\*beta + gamma; **Note:** Saturation is not applied when the output array has the depth CV\_32S. You may even get result of an incorrect sign in the case of overflow.Parameters:src1 - first input array.alpha - weight of the first array elements.src2 - second input array of the same size and channel number as src1.beta - weight of the second array elements.gamma - scalar added to each sum.dst - output array that has the same size and number of channels as the input arrays. can be set to -1, which will be equivalent to src1.depth(). SEE: add, subtract, scaleAdd, Mat::convertTo

#### addWeighted public static void addWeighted([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, double alpha, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, double beta, double gamma, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int dtype) Calculates the weighted sum of two arrays. The function addWeighted calculates the weighted sum of two arrays as follows: \(\texttt{dst} (I)= \texttt{saturate} ( \texttt{src1} (I)\* \texttt{alpha} + \texttt{src2} (I)\* \texttt{beta} + \texttt{gamma} )\) where I is a multi-dimensional index of array elements. In case of multi-channel arrays, each channel is processed independently. The function can be replaced with a matrix expression: dst = src1\*alpha + src2\*beta + gamma; **Note:** Saturation is not applied when the output array has the depth CV\_32S. You may even get result of an incorrect sign in the case of overflow.Parameters:src1 - first input array.alpha - weight of the first array elements.src2 - second input array of the same size and channel number as src1.beta - weight of the second array elements.gamma - scalar added to each sum.dst - output array that has the same size and number of channels as the input arrays.dtype - optional depth of the output array; when both input arrays have the same depth, dtype can be set to -1, which will be equivalent to src1.depth(). SEE: add, subtract, scaleAdd, Mat::convertTo

#### batchDistance public static void batchDistance([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dist, int dtype, [Mat](http://docs.google.com/org/opencv/core/Mat.html) nidx) naive nearest neighbor finder see http://en.wikipedia.org/wiki/Nearest\_neighbor\_search TODO: documentParameters:src1 - automatically generatedsrc2 - automatically generateddist - automatically generateddtype - automatically generatednidx - automatically generated

#### batchDistance public static void batchDistance([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dist, int dtype, [Mat](http://docs.google.com/org/opencv/core/Mat.html) nidx, int normType) naive nearest neighbor finder see http://en.wikipedia.org/wiki/Nearest\_neighbor\_search TODO: documentParameters:src1 - automatically generatedsrc2 - automatically generateddist - automatically generateddtype - automatically generatednidx - automatically generatednormType - automatically generated

#### batchDistance public static void batchDistance([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dist, int dtype, [Mat](http://docs.google.com/org/opencv/core/Mat.html) nidx, int normType, int K) naive nearest neighbor finder see http://en.wikipedia.org/wiki/Nearest\_neighbor\_search TODO: documentParameters:src1 - automatically generatedsrc2 - automatically generateddist - automatically generateddtype - automatically generatednidx - automatically generatednormType - automatically generatedK - automatically generated

#### batchDistance public static void batchDistance([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dist, int dtype, [Mat](http://docs.google.com/org/opencv/core/Mat.html) nidx, int normType, int K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) naive nearest neighbor finder see http://en.wikipedia.org/wiki/Nearest\_neighbor\_search TODO: documentParameters:src1 - automatically generatedsrc2 - automatically generateddist - automatically generateddtype - automatically generatednidx - automatically generatednormType - automatically generatedK - automatically generatedmask - automatically generated

#### batchDistance public static void batchDistance([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dist, int dtype, [Mat](http://docs.google.com/org/opencv/core/Mat.html) nidx, int normType, int K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask, int update) naive nearest neighbor finder see http://en.wikipedia.org/wiki/Nearest\_neighbor\_search TODO: documentParameters:src1 - automatically generatedsrc2 - automatically generateddist - automatically generateddtype - automatically generatednidx - automatically generatednormType - automatically generatedK - automatically generatedmask - automatically generatedupdate - automatically generated

#### batchDistance public static void batchDistance([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dist, int dtype, [Mat](http://docs.google.com/org/opencv/core/Mat.html) nidx, int normType, int K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask, int update, boolean crosscheck) naive nearest neighbor finder see http://en.wikipedia.org/wiki/Nearest\_neighbor\_search TODO: documentParameters:src1 - automatically generatedsrc2 - automatically generateddist - automatically generateddtype - automatically generatednidx - automatically generatednormType - automatically generatedK - automatically generatedmask - automatically generatedupdate - automatically generatedcrosscheck - automatically generated

#### bitwise\_and public static void bitwise\_and([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) computes bitwise conjunction of the two arrays (dst = src1 & src2) Calculates the per-element bit-wise conjunction of two arrays or an array and a scalar. The function cv::bitwise\_and calculates the per-element bit-wise logical conjunction for: Two arrays when src1 and src2 have the same size: \(\texttt{dst} (I) = \texttt{src1} (I) \wedge \texttt{src2} (I) \quad \texttt{if mask} (I) \ne0\) An array and a scalar when src2 is constructed from Scalar or has the same number of elements as src1.channels(): \(\texttt{dst} (I) = \texttt{src1} (I) \wedge \texttt{src2} \quad \texttt{if mask} (I) \ne0\) A scalar and an array when src1 is constructed from Scalar or has the same number of elements as src2.channels(): \(\texttt{dst} (I) = \texttt{src1} \wedge \texttt{src2} (I) \quad \texttt{if mask} (I) \ne0\) In case of floating-point arrays, their machine-specific bit representations (usually IEEE754-compliant) are used for the operation. In case of multi-channel arrays, each channel is processed independently. In the second and third cases above, the scalar is first converted to the array type.Parameters:src1 - first input array or a scalar.src2 - second input array or a scalar.dst - output array that has the same size and type as the input arrays. specifies elements of the output array to be changed.

#### bitwise\_and public static void bitwise\_and([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) computes bitwise conjunction of the two arrays (dst = src1 & src2) Calculates the per-element bit-wise conjunction of two arrays or an array and a scalar. The function cv::bitwise\_and calculates the per-element bit-wise logical conjunction for: Two arrays when src1 and src2 have the same size: \(\texttt{dst} (I) = \texttt{src1} (I) \wedge \texttt{src2} (I) \quad \texttt{if mask} (I) \ne0\) An array and a scalar when src2 is constructed from Scalar or has the same number of elements as src1.channels(): \(\texttt{dst} (I) = \texttt{src1} (I) \wedge \texttt{src2} \quad \texttt{if mask} (I) \ne0\) A scalar and an array when src1 is constructed from Scalar or has the same number of elements as src2.channels(): \(\texttt{dst} (I) = \texttt{src1} \wedge \texttt{src2} (I) \quad \texttt{if mask} (I) \ne0\) In case of floating-point arrays, their machine-specific bit representations (usually IEEE754-compliant) are used for the operation. In case of multi-channel arrays, each channel is processed independently. In the second and third cases above, the scalar is first converted to the array type.Parameters:src1 - first input array or a scalar.src2 - second input array or a scalar.dst - output array that has the same size and type as the input arrays.mask - optional operation mask, 8-bit single channel array, that specifies elements of the output array to be changed.

#### bitwise\_not public static void bitwise\_not([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Inverts every bit of an array. The function cv::bitwise\_not calculates per-element bit-wise inversion of the input array: \(\texttt{dst} (I) = \neg \texttt{src} (I)\) In case of a floating-point input array, its machine-specific bit representation (usually IEEE754-compliant) is used for the operation. In case of multi-channel arrays, each channel is processed independently.Parameters:src - input array.dst - output array that has the same size and type as the input array. specifies elements of the output array to be changed.

#### bitwise\_not public static void bitwise\_not([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) Inverts every bit of an array. The function cv::bitwise\_not calculates per-element bit-wise inversion of the input array: \(\texttt{dst} (I) = \neg \texttt{src} (I)\) In case of a floating-point input array, its machine-specific bit representation (usually IEEE754-compliant) is used for the operation. In case of multi-channel arrays, each channel is processed independently.Parameters:src - input array.dst - output array that has the same size and type as the input array.mask - optional operation mask, 8-bit single channel array, that specifies elements of the output array to be changed.

#### bitwise\_or public static void bitwise\_or([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Calculates the per-element bit-wise disjunction of two arrays or an array and a scalar. The function cv::bitwise\_or calculates the per-element bit-wise logical disjunction for: Two arrays when src1 and src2 have the same size: \(\texttt{dst} (I) = \texttt{src1} (I) \vee \texttt{src2} (I) \quad \texttt{if mask} (I) \ne0\) An array and a scalar when src2 is constructed from Scalar or has the same number of elements as src1.channels(): \(\texttt{dst} (I) = \texttt{src1} (I) \vee \texttt{src2} \quad \texttt{if mask} (I) \ne0\) A scalar and an array when src1 is constructed from Scalar or has the same number of elements as src2.channels(): \(\texttt{dst} (I) = \texttt{src1} \vee \texttt{src2} (I) \quad \texttt{if mask} (I) \ne0\) In case of floating-point arrays, their machine-specific bit representations (usually IEEE754-compliant) are used for the operation. In case of multi-channel arrays, each channel is processed independently. In the second and third cases above, the scalar is first converted to the array type.Parameters:src1 - first input array or a scalar.src2 - second input array or a scalar.dst - output array that has the same size and type as the input arrays. specifies elements of the output array to be changed.

#### bitwise\_or public static void bitwise\_or([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) Calculates the per-element bit-wise disjunction of two arrays or an array and a scalar. The function cv::bitwise\_or calculates the per-element bit-wise logical disjunction for: Two arrays when src1 and src2 have the same size: \(\texttt{dst} (I) = \texttt{src1} (I) \vee \texttt{src2} (I) \quad \texttt{if mask} (I) \ne0\) An array and a scalar when src2 is constructed from Scalar or has the same number of elements as src1.channels(): \(\texttt{dst} (I) = \texttt{src1} (I) \vee \texttt{src2} \quad \texttt{if mask} (I) \ne0\) A scalar and an array when src1 is constructed from Scalar or has the same number of elements as src2.channels(): \(\texttt{dst} (I) = \texttt{src1} \vee \texttt{src2} (I) \quad \texttt{if mask} (I) \ne0\) In case of floating-point arrays, their machine-specific bit representations (usually IEEE754-compliant) are used for the operation. In case of multi-channel arrays, each channel is processed independently. In the second and third cases above, the scalar is first converted to the array type.Parameters:src1 - first input array or a scalar.src2 - second input array or a scalar.dst - output array that has the same size and type as the input arrays.mask - optional operation mask, 8-bit single channel array, that specifies elements of the output array to be changed.

#### bitwise\_xor public static void bitwise\_xor([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Calculates the per-element bit-wise "exclusive or" operation on two arrays or an array and a scalar. The function cv::bitwise\_xor calculates the per-element bit-wise logical "exclusive-or" operation for: Two arrays when src1 and src2 have the same size: \(\texttt{dst} (I) = \texttt{src1} (I) \oplus \texttt{src2} (I) \quad \texttt{if mask} (I) \ne0\) An array and a scalar when src2 is constructed from Scalar or has the same number of elements as src1.channels(): \(\texttt{dst} (I) = \texttt{src1} (I) \oplus \texttt{src2} \quad \texttt{if mask} (I) \ne0\) A scalar and an array when src1 is constructed from Scalar or has the same number of elements as src2.channels(): \(\texttt{dst} (I) = \texttt{src1} \oplus \texttt{src2} (I) \quad \texttt{if mask} (I) \ne0\) In case of floating-point arrays, their machine-specific bit representations (usually IEEE754-compliant) are used for the operation. In case of multi-channel arrays, each channel is processed independently. In the 2nd and 3rd cases above, the scalar is first converted to the array type.Parameters:src1 - first input array or a scalar.src2 - second input array or a scalar.dst - output array that has the same size and type as the input arrays. specifies elements of the output array to be changed.

#### bitwise\_xor public static void bitwise\_xor([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) Calculates the per-element bit-wise "exclusive or" operation on two arrays or an array and a scalar. The function cv::bitwise\_xor calculates the per-element bit-wise logical "exclusive-or" operation for: Two arrays when src1 and src2 have the same size: \(\texttt{dst} (I) = \texttt{src1} (I) \oplus \texttt{src2} (I) \quad \texttt{if mask} (I) \ne0\) An array and a scalar when src2 is constructed from Scalar or has the same number of elements as src1.channels(): \(\texttt{dst} (I) = \texttt{src1} (I) \oplus \texttt{src2} \quad \texttt{if mask} (I) \ne0\) A scalar and an array when src1 is constructed from Scalar or has the same number of elements as src2.channels(): \(\texttt{dst} (I) = \texttt{src1} \oplus \texttt{src2} (I) \quad \texttt{if mask} (I) \ne0\) In case of floating-point arrays, their machine-specific bit representations (usually IEEE754-compliant) are used for the operation. In case of multi-channel arrays, each channel is processed independently. In the 2nd and 3rd cases above, the scalar is first converted to the array type.Parameters:src1 - first input array or a scalar.src2 - second input array or a scalar.dst - output array that has the same size and type as the input arrays.mask - optional operation mask, 8-bit single channel array, that specifies elements of the output array to be changed.

#### borderInterpolate public static int borderInterpolate(int p, int len, int borderType) Computes the source location of an extrapolated pixel. The function computes and returns the coordinate of a donor pixel corresponding to the specified extrapolated pixel when using the specified extrapolation border mode. For example, if you use cv::BORDER\_WRAP mode in the horizontal direction, cv::BORDER\_REFLECT\_101 in the vertical direction and want to compute value of the "virtual" pixel Point(-5, 100) in a floating-point image img , it looks like: float val = img.at<float>(borderInterpolate(100, img.rows, cv::BORDER\_REFLECT\_101), borderInterpolate(-5, img.cols, cv::BORDER\_WRAP)); Normally, the function is not called directly. It is used inside filtering functions and also in copyMakeBorder.Parameters:p - 0-based coordinate of the extrapolated pixel along one of the axes, likely <0 or >= lenlen - Length of the array along the corresponding axis.borderType - Border type, one of the #BorderTypes, except for #BORDER\_TRANSPARENT and #BORDER\_ISOLATED . When borderType==#BORDER\_CONSTANT , the function always returns -1, regardless of p and len. SEE: copyMakeBorder Returns:automatically generated

#### calcCovarMatrix public static void calcCovarMatrix([Mat](http://docs.google.com/org/opencv/core/Mat.html) samples, [Mat](http://docs.google.com/org/opencv/core/Mat.html) covar, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mean, int flags) **Note:** use #COVAR\_ROWS or #COVAR\_COLS flagParameters:samples - samples stored as rows/columns of a single matrix.covar - output covariance matrix of the type ctype and square size.mean - input or output (depending on the flags) array as the average value of the input vectors.flags - operation flags as a combination of #CovarFlags

#### calcCovarMatrix public static void calcCovarMatrix([Mat](http://docs.google.com/org/opencv/core/Mat.html) samples, [Mat](http://docs.google.com/org/opencv/core/Mat.html) covar, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mean, int flags, int ctype) **Note:** use #COVAR\_ROWS or #COVAR\_COLS flagParameters:samples - samples stored as rows/columns of a single matrix.covar - output covariance matrix of the type ctype and square size.mean - input or output (depending on the flags) array as the average value of the input vectors.flags - operation flags as a combination of #CovarFlagsctype - type of the matrixl; it equals 'CV\_64F' by default.

#### cartToPolar public static void cartToPolar([Mat](http://docs.google.com/org/opencv/core/Mat.html) x, [Mat](http://docs.google.com/org/opencv/core/Mat.html) y, [Mat](http://docs.google.com/org/opencv/core/Mat.html) magnitude, [Mat](http://docs.google.com/org/opencv/core/Mat.html) angle) Calculates the magnitude and angle of 2D vectors. The function cv::cartToPolar calculates either the magnitude, angle, or both for every 2D vector (x(I),y(I)): \(\begin{array}{l} \texttt{magnitude} (I)= \sqrt{\texttt{x}(I)^2+\texttt{y}(I)^2} , \\ \texttt{angle} (I)= \texttt{atan2} ( \texttt{y} (I), \texttt{x} (I))[ \cdot180 / \pi ] \end{array}\) The angles are calculated with accuracy about 0.3 degrees. For the point (0,0), the angle is set to 0.Parameters:x - array of x-coordinates; this must be a single-precision or double-precision floating-point array.y - array of y-coordinates, that must have the same size and same type as x.magnitude - output array of magnitudes of the same size and type as x.angle - output array of angles that has the same size and type as x; the angles are measured in radians (from 0 to 2\\*Pi) or in degrees (0 to 360 degrees). in radians (which is by default), or in degrees. SEE: Sobel, Scharr

#### cartToPolar public static void cartToPolar([Mat](http://docs.google.com/org/opencv/core/Mat.html) x, [Mat](http://docs.google.com/org/opencv/core/Mat.html) y, [Mat](http://docs.google.com/org/opencv/core/Mat.html) magnitude, [Mat](http://docs.google.com/org/opencv/core/Mat.html) angle, boolean angleInDegrees) Calculates the magnitude and angle of 2D vectors. The function cv::cartToPolar calculates either the magnitude, angle, or both for every 2D vector (x(I),y(I)): \(\begin{array}{l} \texttt{magnitude} (I)= \sqrt{\texttt{x}(I)^2+\texttt{y}(I)^2} , \\ \texttt{angle} (I)= \texttt{atan2} ( \texttt{y} (I), \texttt{x} (I))[ \cdot180 / \pi ] \end{array}\) The angles are calculated with accuracy about 0.3 degrees. For the point (0,0), the angle is set to 0.Parameters:x - array of x-coordinates; this must be a single-precision or double-precision floating-point array.y - array of y-coordinates, that must have the same size and same type as x.magnitude - output array of magnitudes of the same size and type as x.angle - output array of angles that has the same size and type as x; the angles are measured in radians (from 0 to 2\\*Pi) or in degrees (0 to 360 degrees).angleInDegrees - a flag, indicating whether the angles are measured in radians (which is by default), or in degrees. SEE: Sobel, Scharr

#### checkRange public static boolean checkRange([Mat](http://docs.google.com/org/opencv/core/Mat.html) a)

Checks every element of an input array for invalid values. The function cv::checkRange checks that every array element is neither NaN nor infinite. When minVal >

* + - * DBL\_MAX and maxVal < DBL\_MAX, the function also checks that each value is between minVal and maxVal. In case of multi-channel arrays, each channel is processed independently. If some values are out of range, position of the first outlier is stored in pos (when pos != NULL). Then, the function either returns false (when quiet=true) or throws an exception.

Parameters:a - input array. are out of range or they throw an exception. elements.Returns:automatically generated

#### checkRange public static boolean checkRange([Mat](http://docs.google.com/org/opencv/core/Mat.html) a, boolean quiet)

Checks every element of an input array for invalid values. The function cv::checkRange checks that every array element is neither NaN nor infinite. When minVal >

* + - DBL\_MAX and maxVal < DBL\_MAX, the function also checks that each value is between minVal and maxVal. In case of multi-channel arrays, each channel is processed independently. If some values are out of range, position of the first outlier is stored in pos (when pos != NULL). Then, the function either returns false (when quiet=true) or throws an exception.

Parameters:a - input array.quiet - a flag, indicating whether the functions quietly return false when the array elements are out of range or they throw an exception. elements.Returns:automatically generated

#### checkRange public static boolean checkRange([Mat](http://docs.google.com/org/opencv/core/Mat.html) a, boolean quiet, double minVal)

Checks every element of an input array for invalid values. The function cv::checkRange checks that every array element is neither NaN nor infinite. When minVal >

* + DBL\_MAX and maxVal < DBL\_MAX, the function also checks that each value is between minVal and maxVal. In case of multi-channel arrays, each channel is processed independently. If some values are out of range, position of the first outlier is stored in pos (when pos != NULL). Then, the function either returns false (when quiet=true) or throws an exception.

Parameters:a - input array.quiet - a flag, indicating whether the functions quietly return false when the array elements are out of range or they throw an exception. elements.minVal - inclusive lower boundary of valid values range.Returns:automatically generated

#### checkRange public static boolean checkRange([Mat](http://docs.google.com/org/opencv/core/Mat.html) a, boolean quiet, double minVal, double maxVal)

Checks every element of an input array for invalid values. The function cv::checkRange checks that every array element is neither NaN nor infinite. When minVal >

* + DBL\_MAX and maxVal < DBL\_MAX, the function also checks that each value is between minVal and maxVal. In case of multi-channel arrays, each channel is processed independently. If some values are out of range, position of the first outlier is stored in pos (when pos != NULL). Then, the function either returns false (when quiet=true) or throws an exception.

Parameters:a - input array.quiet - a flag, indicating whether the functions quietly return false when the array elements are out of range or they throw an exception. elements.minVal - inclusive lower boundary of valid values range.maxVal - exclusive upper boundary of valid values range.Returns:automatically generated

#### compare public static void compare([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int cmpop) Performs the per-element comparison of two arrays or an array and scalar value. The function compares: Elements of two arrays when src1 and src2 have the same size: \(\texttt{dst} (I) = \texttt{src1} (I) \,\texttt{cmpop}\, \texttt{src2} (I)\) Elements of src1 with a scalar src2 when src2 is constructed from Scalar or has a single element: \(\texttt{dst} (I) = \texttt{src1}(I) \,\texttt{cmpop}\, \texttt{src2}\) src1 with elements of src2 when src1 is constructed from Scalar or has a single element: \(\texttt{dst} (I) = \texttt{src1} \,\texttt{cmpop}\, \texttt{src2} (I)\) When the comparison result is true, the corresponding element of output array is set to 255. The comparison operations can be replaced with the equivalent matrix expressions: Mat dst1 = src1 >= src2; Mat dst2 = src1 < 8; ...Parameters:src1 - first input array or a scalar; when it is an array, it must have a single channel.src2 - second input array or a scalar; when it is an array, it must have a single channel.dst - output array of type ref CV\_8U that has the same size and the same number of channels as the input arrays.cmpop - a flag, that specifies correspondence between the arrays (cv::CmpTypes) SEE: checkRange, min, max, threshold

#### compare public static void compare([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int cmpop)

#### completeSymm public static void completeSymm([Mat](http://docs.google.com/org/opencv/core/Mat.html) m)

Copies the lower or the upper half of a square matrix to its another half. The function cv::completeSymm copies the lower or the upper half of a square matrix to its another half. The matrix diagonal remains unchanged:

* + \(\texttt{m}\_{ij}=\texttt{m}\_{ji}\) for \(i > j\) if lowerToUpper=false
  + \(\texttt{m}\_{ij}=\texttt{m}\_{ji}\) for \(i < j\) if lowerToUpper=true

Parameters:m - input-output floating-point square matrix. the upper half. Otherwise, the upper half is copied to the lower half. SEE: flip, transpose

#### completeSymm public static void completeSymm([Mat](http://docs.google.com/org/opencv/core/Mat.html) m, boolean lowerToUpper)

Copies the lower or the upper half of a square matrix to its another half. The function cv::completeSymm copies the lower or the upper half of a square matrix to its another half. The matrix diagonal remains unchanged:

* + \(\texttt{m}\_{ij}=\texttt{m}\_{ji}\) for \(i > j\) if lowerToUpper=false
  + \(\texttt{m}\_{ij}=\texttt{m}\_{ji}\) for \(i < j\) if lowerToUpper=true

Parameters:m - input-output floating-point square matrix.lowerToUpper - operation flag; if true, the lower half is copied to the upper half. Otherwise, the upper half is copied to the lower half. SEE: flip, transpose

#### convertFp16 public static void convertFp16([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Converts an array to half precision floating number. This function converts FP32 (single precision floating point) from/to FP16 (half precision floating point). CV\_16S format is used to represent FP16 data. There are two use modes (src -> dst): CV\_32F -> CV\_16S and CV\_16S -> CV\_32F. The input array has to have type of CV\_32F or CV\_16S to represent the bit depth. If the input array is neither of them, the function will raise an error. The format of half precision floating point is defined in IEEE 754-2008.Parameters:src - input array.dst - output array.

#### convertScaleAbs public static void convertScaleAbs([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Scales, calculates absolute values, and converts the result to 8-bit. On each element of the input array, the function convertScaleAbs performs three operations sequentially: scaling, taking an absolute value, conversion to an unsigned 8-bit type: \(\texttt{dst} (I)= \texttt{saturate\\_cast<uchar>} (| \texttt{src} (I)\* \texttt{alpha} + \texttt{beta} |)\) In case of multi-channel arrays, the function processes each channel independently. When the output is not 8-bit, the operation can be emulated by calling the Mat::convertTo method (or by using matrix expressions) and then by calculating an absolute value of the result. For example: Mat\_<float> A(30,30); randu(A, Scalar(-100), Scalar(100)); Mat\_<float> B = A\*5 + 3; B = abs(B); // Mat\_<float> B = abs(A\*5+3) will also do the job, // but it will allocate a temporary matrixParameters:src - input array.dst - output array. SEE: Mat::convertTo, cv::abs(const Mat&)

#### convertScaleAbs public static void convertScaleAbs([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double alpha) Scales, calculates absolute values, and converts the result to 8-bit. On each element of the input array, the function convertScaleAbs performs three operations sequentially: scaling, taking an absolute value, conversion to an unsigned 8-bit type: \(\texttt{dst} (I)= \texttt{saturate\\_cast<uchar>} (| \texttt{src} (I)\* \texttt{alpha} + \texttt{beta} |)\) In case of multi-channel arrays, the function processes each channel independently. When the output is not 8-bit, the operation can be emulated by calling the Mat::convertTo method (or by using matrix expressions) and then by calculating an absolute value of the result. For example: Mat\_<float> A(30,30); randu(A, Scalar(-100), Scalar(100)); Mat\_<float> B = A\*5 + 3; B = abs(B); // Mat\_<float> B = abs(A\*5+3) will also do the job, // but it will allocate a temporary matrixParameters:src - input array.dst - output array.alpha - optional scale factor. SEE: Mat::convertTo, cv::abs(const Mat&)

#### convertScaleAbs public static void convertScaleAbs([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double alpha, double beta) Scales, calculates absolute values, and converts the result to 8-bit. On each element of the input array, the function convertScaleAbs performs three operations sequentially: scaling, taking an absolute value, conversion to an unsigned 8-bit type: \(\texttt{dst} (I)= \texttt{saturate\\_cast<uchar>} (| \texttt{src} (I)\* \texttt{alpha} + \texttt{beta} |)\) In case of multi-channel arrays, the function processes each channel independently. When the output is not 8-bit, the operation can be emulated by calling the Mat::convertTo method (or by using matrix expressions) and then by calculating an absolute value of the result. For example: Mat\_<float> A(30,30); randu(A, Scalar(-100), Scalar(100)); Mat\_<float> B = A\*5 + 3; B = abs(B); // Mat\_<float> B = abs(A\*5+3) will also do the job, // but it will allocate a temporary matrixParameters:src - input array.dst - output array.alpha - optional scale factor.beta - optional delta added to the scaled values. SEE: Mat::convertTo, cv::abs(const Mat&)

#### copyMakeBorder public static void copyMakeBorder([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int top, int bottom, int left, int right, int borderType) Forms a border around an image. The function copies the source image into the middle of the destination image. The areas to the left, to the right, above and below the copied source image will be filled with extrapolated pixels. This is not what filtering functions based on it do (they extrapolate pixels on-fly), but what other more complex functions, including your own, may do to simplify image boundary handling. The function supports the mode when src is already in the middle of dst . In this case, the function does not copy src itself but simply constructs the border, for example: // let border be the same in all directions int border=2; // constructs a larger image to fit both the image and the border Mat gray\_buf(rgb.rows + border\*2, rgb.cols + border\*2, rgb.depth()); // select the middle part of it w/o copying data Mat gray(gray\_canvas, Rect(border, border, rgb.cols, rgb.rows)); // convert image from RGB to grayscale cvtColor(rgb, gray, COLOR\_RGB2GRAY); // form a border in-place copyMakeBorder(gray, gray\_buf, border, border, border, border, BORDER\_REPLICATE); // now do some custom filtering ... ... **Note:** When the source image is a part (ROI) of a bigger image, the function will try to use the pixels outside of the ROI to form a border. To disable this feature and always do extrapolation, as if src was not a ROI, use borderType | #BORDER\_ISOLATED.Parameters:src - Source image.dst - Destination image of the same type as src and the size Size(src.cols+left+right, src.rows+top+bottom) .top - the top pixelsbottom - the bottom pixelsleft - the left pixelsright - Parameter specifying how many pixels in each direction from the source image rectangle to extrapolate. For example, top=1, bottom=1, left=1, right=1 mean that 1 pixel-wide border needs to be built.borderType - Border type. See borderInterpolate for details. SEE: borderInterpolate

#### copyMakeBorder public static void copyMakeBorder([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int top, int bottom, int left, int right, int borderType, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) value) Forms a border around an image. The function copies the source image into the middle of the destination image. The areas to the left, to the right, above and below the copied source image will be filled with extrapolated pixels. This is not what filtering functions based on it do (they extrapolate pixels on-fly), but what other more complex functions, including your own, may do to simplify image boundary handling. The function supports the mode when src is already in the middle of dst . In this case, the function does not copy src itself but simply constructs the border, for example: // let border be the same in all directions int border=2; // constructs a larger image to fit both the image and the border Mat gray\_buf(rgb.rows + border\*2, rgb.cols + border\*2, rgb.depth()); // select the middle part of it w/o copying data Mat gray(gray\_canvas, Rect(border, border, rgb.cols, rgb.rows)); // convert image from RGB to grayscale cvtColor(rgb, gray, COLOR\_RGB2GRAY); // form a border in-place copyMakeBorder(gray, gray\_buf, border, border, border, border, BORDER\_REPLICATE); // now do some custom filtering ... ... **Note:** When the source image is a part (ROI) of a bigger image, the function will try to use the pixels outside of the ROI to form a border. To disable this feature and always do extrapolation, as if src was not a ROI, use borderType | #BORDER\_ISOLATED.Parameters:src - Source image.dst - Destination image of the same type as src and the size Size(src.cols+left+right, src.rows+top+bottom) .top - the top pixelsbottom - the bottom pixelsleft - the left pixelsright - Parameter specifying how many pixels in each direction from the source image rectangle to extrapolate. For example, top=1, bottom=1, left=1, right=1 mean that 1 pixel-wide border needs to be built.borderType - Border type. See borderInterpolate for details.value - Border value if borderType==BORDER\_CONSTANT . SEE: borderInterpolate

#### countNonZero public static int countNonZero([Mat](http://docs.google.com/org/opencv/core/Mat.html) src) Counts non-zero array elements. The function returns the number of non-zero elements in src : \(\sum \_{I: \; \texttt{src} (I) \ne0 } 1\)Parameters:src - single-channel array. SEE: mean, meanStdDev, norm, minMaxLoc, calcCovarMatrix Returns:automatically generated

#### cubeRoot public static float cubeRoot(float val) Computes the cube root of an argument. The function cubeRoot computes \(\sqrt[3]{\texttt{val}}\). Negative arguments are handled correctly. NaN and Inf are not handled. The accuracy approaches the maximum possible accuracy for single-precision data.Parameters:val - A function argument. Returns:automatically generated

#### dct public static void dct([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst)

Performs a forward or inverse discrete Cosine transform of 1D or 2D array. The function cv::dct performs a forward or inverse discrete Cosine transform (DCT) of a 1D or 2D floating-point array:

* + Forward Cosine transform of a 1D vector of N elements: \(Y = C^{(N)} \cdot X\) where \(C^{(N)}\_{jk}= \sqrt{\alpha\_j/N} \cos \left ( \frac{\pi(2k+1)j}{2N} \right )\) and \(\alpha\_0=1\), \(\alpha\_j=2\) for \*j > 0\*.
  + Inverse Cosine transform of a 1D vector of N elements: \(X = \left (C^{(N)} \right )^{-1} \cdot Y = \left (C^{(N)} \right )^T \cdot Y\) (since \(C^{(N)}\) is an orthogonal matrix, \(C^{(N)} \cdot \left(C^{(N)}\right)^T = I\) )
  + Forward 2D Cosine transform of M x N matrix: \(Y = C^{(N)} \cdot X \cdot \left (C^{(N)} \right )^T\)
  + Inverse 2D Cosine transform of M x N matrix: \(X = \left (C^{(N)} \right )^T \cdot X \cdot C^{(N)}\)

The function chooses the mode of operation by looking at the flags and size of the input array:

* + If (flags & #DCT\_INVERSE) == 0 , the function does a forward 1D or 2D transform. Otherwise, it is an inverse 1D or 2D transform.
  + If (flags & #DCT\_ROWS) != 0 , the function performs a 1D transform of each row.
  + If the array is a single column or a single row, the function performs a 1D transform.
  + If none of the above is true, the function performs a 2D transform.

**Note:** Currently dct supports even-size arrays (2, 4, 6 ...). For data analysis and approximation, you can pad the array when necessary. Also, the function performance depends very much, and not monotonically, on the array size (see getOptimalDFTSize ). In the current implementation DCT of a vector of size N is calculated via DFT of a vector of size N/2 . Thus, the optimal DCT size N1 >= N can be calculated as: size\_t getOptimalDCTSize(size\_t N) { return 2\*getOptimalDFTSize((N+1)/2); } N1 = getOptimalDCTSize(N);Parameters:src - input floating-point array.dst - output array of the same size and type as src . SEE: dft , getOptimalDFTSize , idct

#### dct public static void dct([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int flags)

Performs a forward or inverse discrete Cosine transform of 1D or 2D array. The function cv::dct performs a forward or inverse discrete Cosine transform (DCT) of a 1D or 2D floating-point array:

* + Forward Cosine transform of a 1D vector of N elements: \(Y = C^{(N)} \cdot X\) where \(C^{(N)}\_{jk}= \sqrt{\alpha\_j/N} \cos \left ( \frac{\pi(2k+1)j}{2N} \right )\) and \(\alpha\_0=1\), \(\alpha\_j=2\) for \*j > 0\*.
  + Inverse Cosine transform of a 1D vector of N elements: \(X = \left (C^{(N)} \right )^{-1} \cdot Y = \left (C^{(N)} \right )^T \cdot Y\) (since \(C^{(N)}\) is an orthogonal matrix, \(C^{(N)} \cdot \left(C^{(N)}\right)^T = I\) )
  + Forward 2D Cosine transform of M x N matrix: \(Y = C^{(N)} \cdot X \cdot \left (C^{(N)} \right )^T\)
  + Inverse 2D Cosine transform of M x N matrix: \(X = \left (C^{(N)} \right )^T \cdot X \cdot C^{(N)}\)

The function chooses the mode of operation by looking at the flags and size of the input array:

* + If (flags & #DCT\_INVERSE) == 0 , the function does a forward 1D or 2D transform. Otherwise, it is an inverse 1D or 2D transform.
  + If (flags & #DCT\_ROWS) != 0 , the function performs a 1D transform of each row.
  + If the array is a single column or a single row, the function performs a 1D transform.
  + If none of the above is true, the function performs a 2D transform.

**Note:** Currently dct supports even-size arrays (2, 4, 6 ...). For data analysis and approximation, you can pad the array when necessary. Also, the function performance depends very much, and not monotonically, on the array size (see getOptimalDFTSize ). In the current implementation DCT of a vector of size N is calculated via DFT of a vector of size N/2 . Thus, the optimal DCT size N1 >= N can be calculated as: size\_t getOptimalDCTSize(size\_t N) { return 2\*getOptimalDFTSize((N+1)/2); } N1 = getOptimalDCTSize(N);Parameters:src - input floating-point array.dst - output array of the same size and type as src .flags - transformation flags as a combination of cv::DftFlags (DCT\_\*) SEE: dft , getOptimalDFTSize , idct

#### determinant public static double determinant([Mat](http://docs.google.com/org/opencv/core/Mat.html) mtx) Returns the determinant of a square floating-point matrix. The function cv::determinant calculates and returns the determinant of the specified matrix. For small matrices ( mtx.cols=mtx.rows<=3 ), the direct method is used. For larger matrices, the function uses LU factorization with partial pivoting. For symmetric positively-determined matrices, it is also possible to use eigen decomposition to calculate the determinant.Parameters:mtx - input matrix that must have CV\_32FC1 or CV\_64FC1 type and square size. SEE: trace, invert, solve, eigen, REF: MatrixExpressions Returns:automatically generated

#### dft public static void dft([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst)

Performs a forward or inverse Discrete Fourier transform of a 1D or 2D floating-point array. The function cv::dft performs one of the following:

* + Forward the Fourier transform of a 1D vector of N elements: \(Y = F^{(N)} \cdot X,\) where \(F^{(N)}\_{jk}=\exp(-2\pi i j k/N)\) and \(i=\sqrt{-1}\)
  + Inverse the Fourier transform of a 1D vector of N elements: \(\begin{array}{l} X'= \left (F^{(N)} \right )^{-1} \cdot Y = \left (F^{(N)} \right )^\* \cdot y \\ X = (1/N) \cdot X, \end{array}\) where \(F^\*=\left(\textrm{Re}(F^{(N)})-\textrm{Im}(F^{(N)})\right)^T\)
  + Forward the 2D Fourier transform of a M x N matrix: \(Y = F^{(M)} \cdot X \cdot F^{(N)}\)
  + Inverse the 2D Fourier transform of a M x N matrix: \(\begin{array}{l} X'= \left (F^{(M)} \right )^\* \cdot Y \cdot \left (F^{(N)} \right )^\* \\ X = \frac{1}{M \cdot N} \cdot X' \end{array}\)

In case of real (single-channel) data, the output spectrum of the forward Fourier transform or input spectrum of the inverse Fourier transform can be represented in a packed format called \*CCS\* (complex-conjugate-symmetrical). It was borrowed from IPL (Intel\\* Image Processing Library). Here is how 2D \*CCS\* spectrum looks: \(\begin{bmatrix} Re Y\_{0,0} & Re Y\_{0,1} & Im Y\_{0,1} & Re Y\_{0,2} & Im Y\_{0,2} & \cdots & Re Y\_{0,N/2-1} & Im Y\_{0,N/2-1} & Re Y\_{0,N/2} \\ Re Y\_{1,0} & Re Y\_{1,1} & Im Y\_{1,1} & Re Y\_{1,2} & Im Y\_{1,2} & \cdots & Re Y\_{1,N/2-1} & Im Y\_{1,N/2-1} & Re Y\_{1,N/2} \\ Im Y\_{1,0} & Re Y\_{2,1} & Im Y\_{2,1} & Re Y\_{2,2} & Im Y\_{2,2} & \cdots & Re Y\_{2,N/2-1} & Im Y\_{2,N/2-1} & Im Y\_{1,N/2} \\ \hdotsfor{9} \\ Re Y\_{M/2-1,0} & Re Y\_{M-3,1} & Im Y\_{M-3,1} & \hdotsfor{3} & Re Y\_{M-3,N/2-1} & Im Y\_{M-3,N/2-1}& Re Y\_{M/2-1,N/2} \\ Im Y\_{M/2-1,0} & Re Y\_{M-2,1} & Im Y\_{M-2,1} & \hdotsfor{3} & Re Y\_{M-2,N/2-1} & Im Y\_{M-2,N/2-1}& Im Y\_{M/2-1,N/2} \\ Re Y\_{M/2,0} & Re Y\_{M-1,1} & Im Y\_{M-1,1} & \hdotsfor{3} & Re Y\_{M-1,N/2-1} & Im Y\_{M-1,N/2-1}& Re Y\_{M/2,N/2} \end{bmatrix}\) In case of 1D transform of a real vector, the output looks like the first row of the matrix above. So, the function chooses an operation mode depending on the flags and size of the input array:

* + If #DFT\_ROWS is set or the input array has a single row or single column, the function performs a 1D forward or inverse transform of each row of a matrix when #DFT\_ROWS is set. Otherwise, it performs a 2D transform.
  + If the input array is real and #DFT\_INVERSE is not set, the function performs a forward 1D or 2D transform:
    - When #DFT\_COMPLEX\_OUTPUT is set, the output is a complex matrix of the same size as input.
    - When #DFT\_COMPLEX\_OUTPUT is not set, the output is a real matrix of the same size as input. In case of 2D transform, it uses the packed format as shown above. In case of a single 1D transform, it looks like the first row of the matrix above. In case of multiple 1D transforms (when using the #DFT\_ROWS flag), each row of the output matrix looks like the first row of the matrix above.
  + If the input array is complex and either #DFT\_INVERSE or #DFT\_REAL\_OUTPUT are not set, the output is a complex array of the same size as input. The function performs a forward or inverse 1D or 2D transform of the whole input array or each row of the input array independently, depending on the flags DFT\_INVERSE and DFT\_ROWS.
  + When #DFT\_INVERSE is set and the input array is real, or it is complex but #DFT\_REAL\_OUTPUT is set, the output is a real array of the same size as input. The function performs a 1D or 2D inverse transformation of the whole input array or each individual row, depending on the flags #DFT\_INVERSE and #DFT\_ROWS.

If #DFT\_SCALE is set, the scaling is done after the transformation. Unlike dct , the function supports arrays of arbitrary size. But only those arrays are processed efficiently, whose sizes can be factorized in a product of small prime numbers (2, 3, and 5 in the current implementation). Such an efficient DFT size can be calculated using the getOptimalDFTSize method. The sample below illustrates how to calculate a DFT-based convolution of two 2D real arrays: void convolveDFT(InputArray A, InputArray B, OutputArray C) { // reallocate the output array if needed C.create(abs(A.rows - B.rows)+1, abs(A.cols - B.cols)+1, A.type()); Size dftSize; // calculate the size of DFT transform dftSize.width = getOptimalDFTSize(A.cols + B.cols - 1); dftSize.height = getOptimalDFTSize(A.rows + B.rows - 1); // allocate temporary buffers and initialize them with 0's Mat tempA(dftSize, A.type(), Scalar::all(0)); Mat tempB(dftSize, B.type(), Scalar::all(0)); // copy A and B to the top-left corners of tempA and tempB, respectively Mat roiA(tempA, Rect(0,0,A.cols,A.rows)); A.copyTo(roiA); Mat roiB(tempB, Rect(0,0,B.cols,B.rows)); B.copyTo(roiB); // now transform the padded A & B in-place; // use "nonzeroRows" hint for faster processing dft(tempA, tempA, 0, A.rows); dft(tempB, tempB, 0, B.rows); // multiply the spectrums; // the function handles packed spectrum representations well mulSpectrums(tempA, tempB, tempA); // transform the product back from the frequency domain. // Even though all the result rows will be non-zero, // you need only the first C.rows of them, and thus you // pass nonzeroRows == C.rows dft(tempA, tempA, DFT\_INVERSE + DFT\_SCALE, C.rows); // now copy the result back to C. tempA(Rect(0, 0, C.cols, C.rows)).copyTo(C); // all the temporary buffers will be deallocated automatically } To optimize this sample, consider the following approaches:

* + Since nonzeroRows != 0 is passed to the forward transform calls and since A and B are copied to the top-left corners of tempA and tempB, respectively, it is not necessary to clear the whole tempA and tempB. It is only necessary to clear the tempA.cols - A.cols ( tempB.cols - B.cols) rightmost columns of the matrices.
  + This DFT-based convolution does not have to be applied to the whole big arrays, especially if B is significantly smaller than A or vice versa. Instead, you can calculate convolution by parts. To do this, you need to split the output array C into multiple tiles. For each tile, estimate which parts of A and B are required to calculate convolution in this tile. If the tiles in C are too small, the speed will decrease a lot because of repeated work. In the ultimate case, when each tile in C is a single pixel, the algorithm becomes equivalent to the naive convolution algorithm. If the tiles are too big, the temporary arrays tempA and tempB become too big and there is also a slowdown because of bad cache locality. So, there is an optimal tile size somewhere in the middle.
  + If different tiles in C can be calculated in parallel and, thus, the convolution is done by parts, the loop can be threaded.

All of the above improvements have been implemented in #matchTemplate and #filter2D . Therefore, by using them, you can get the performance even better than with the above theoretically optimal implementation. Though, those two functions actually calculate cross-correlation, not convolution, so you need to "flip" the second convolution operand B vertically and horizontally using flip . **Note:**

* + An example using the discrete fourier transform can be found at opencv\_source\_code/samples/cpp/dft.cpp
  + (Python) An example using the dft functionality to perform Wiener deconvolution can be found at opencv\_source/samples/python/deconvolution.py
  + (Python) An example rearranging the quadrants of a Fourier image can be found at opencv\_source/samples/python/dft.py

Parameters:src - input array that could be real or complex.dst - output array whose size and type depends on the flags . nonzeroRows rows of the input array (#DFT\_INVERSE is not set) or only the first nonzeroRows of the output array (#DFT\_INVERSE is set) contain non-zeros, thus, the function can handle the rest of the rows more efficiently and save some time; this technique is very useful for calculating array cross-correlation or convolution using DFT. SEE: dct , getOptimalDFTSize , mulSpectrums, filter2D , matchTemplate , flip , cartToPolar , magnitude , phase

#### dft public static void dft([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int flags)

Performs a forward or inverse Discrete Fourier transform of a 1D or 2D floating-point array. The function cv::dft performs one of the following:

* + Forward the Fourier transform of a 1D vector of N elements: \(Y = F^{(N)} \cdot X,\) where \(F^{(N)}\_{jk}=\exp(-2\pi i j k/N)\) and \(i=\sqrt{-1}\)
  + Inverse the Fourier transform of a 1D vector of N elements: \(\begin{array}{l} X'= \left (F^{(N)} \right )^{-1} \cdot Y = \left (F^{(N)} \right )^\* \cdot y \\ X = (1/N) \cdot X, \end{array}\) where \(F^\*=\left(\textrm{Re}(F^{(N)})-\textrm{Im}(F^{(N)})\right)^T\)
  + Forward the 2D Fourier transform of a M x N matrix: \(Y = F^{(M)} \cdot X \cdot F^{(N)}\)
  + Inverse the 2D Fourier transform of a M x N matrix: \(\begin{array}{l} X'= \left (F^{(M)} \right )^\* \cdot Y \cdot \left (F^{(N)} \right )^\* \\ X = \frac{1}{M \cdot N} \cdot X' \end{array}\)

In case of real (single-channel) data, the output spectrum of the forward Fourier transform or input spectrum of the inverse Fourier transform can be represented in a packed format called \*CCS\* (complex-conjugate-symmetrical). It was borrowed from IPL (Intel\\* Image Processing Library). Here is how 2D \*CCS\* spectrum looks: \(\begin{bmatrix} Re Y\_{0,0} & Re Y\_{0,1} & Im Y\_{0,1} & Re Y\_{0,2} & Im Y\_{0,2} & \cdots & Re Y\_{0,N/2-1} & Im Y\_{0,N/2-1} & Re Y\_{0,N/2} \\ Re Y\_{1,0} & Re Y\_{1,1} & Im Y\_{1,1} & Re Y\_{1,2} & Im Y\_{1,2} & \cdots & Re Y\_{1,N/2-1} & Im Y\_{1,N/2-1} & Re Y\_{1,N/2} \\ Im Y\_{1,0} & Re Y\_{2,1} & Im Y\_{2,1} & Re Y\_{2,2} & Im Y\_{2,2} & \cdots & Re Y\_{2,N/2-1} & Im Y\_{2,N/2-1} & Im Y\_{1,N/2} \\ \hdotsfor{9} \\ Re Y\_{M/2-1,0} & Re Y\_{M-3,1} & Im Y\_{M-3,1} & \hdotsfor{3} & Re Y\_{M-3,N/2-1} & Im Y\_{M-3,N/2-1}& Re Y\_{M/2-1,N/2} \\ Im Y\_{M/2-1,0} & Re Y\_{M-2,1} & Im Y\_{M-2,1} & \hdotsfor{3} & Re Y\_{M-2,N/2-1} & Im Y\_{M-2,N/2-1}& Im Y\_{M/2-1,N/2} \\ Re Y\_{M/2,0} & Re Y\_{M-1,1} & Im Y\_{M-1,1} & \hdotsfor{3} & Re Y\_{M-1,N/2-1} & Im Y\_{M-1,N/2-1}& Re Y\_{M/2,N/2} \end{bmatrix}\) In case of 1D transform of a real vector, the output looks like the first row of the matrix above. So, the function chooses an operation mode depending on the flags and size of the input array:

* + If #DFT\_ROWS is set or the input array has a single row or single column, the function performs a 1D forward or inverse transform of each row of a matrix when #DFT\_ROWS is set. Otherwise, it performs a 2D transform.
  + If the input array is real and #DFT\_INVERSE is not set, the function performs a forward 1D or 2D transform:
    - When #DFT\_COMPLEX\_OUTPUT is set, the output is a complex matrix of the same size as input.
    - When #DFT\_COMPLEX\_OUTPUT is not set, the output is a real matrix of the same size as input. In case of 2D transform, it uses the packed format as shown above. In case of a single 1D transform, it looks like the first row of the matrix above. In case of multiple 1D transforms (when using the #DFT\_ROWS flag), each row of the output matrix looks like the first row of the matrix above.
  + If the input array is complex and either #DFT\_INVERSE or #DFT\_REAL\_OUTPUT are not set, the output is a complex array of the same size as input. The function performs a forward or inverse 1D or 2D transform of the whole input array or each row of the input array independently, depending on the flags DFT\_INVERSE and DFT\_ROWS.
  + When #DFT\_INVERSE is set and the input array is real, or it is complex but #DFT\_REAL\_OUTPUT is set, the output is a real array of the same size as input. The function performs a 1D or 2D inverse transformation of the whole input array or each individual row, depending on the flags #DFT\_INVERSE and #DFT\_ROWS.

If #DFT\_SCALE is set, the scaling is done after the transformation. Unlike dct , the function supports arrays of arbitrary size. But only those arrays are processed efficiently, whose sizes can be factorized in a product of small prime numbers (2, 3, and 5 in the current implementation). Such an efficient DFT size can be calculated using the getOptimalDFTSize method. The sample below illustrates how to calculate a DFT-based convolution of two 2D real arrays: void convolveDFT(InputArray A, InputArray B, OutputArray C) { // reallocate the output array if needed C.create(abs(A.rows - B.rows)+1, abs(A.cols - B.cols)+1, A.type()); Size dftSize; // calculate the size of DFT transform dftSize.width = getOptimalDFTSize(A.cols + B.cols - 1); dftSize.height = getOptimalDFTSize(A.rows + B.rows - 1); // allocate temporary buffers and initialize them with 0's Mat tempA(dftSize, A.type(), Scalar::all(0)); Mat tempB(dftSize, B.type(), Scalar::all(0)); // copy A and B to the top-left corners of tempA and tempB, respectively Mat roiA(tempA, Rect(0,0,A.cols,A.rows)); A.copyTo(roiA); Mat roiB(tempB, Rect(0,0,B.cols,B.rows)); B.copyTo(roiB); // now transform the padded A & B in-place; // use "nonzeroRows" hint for faster processing dft(tempA, tempA, 0, A.rows); dft(tempB, tempB, 0, B.rows); // multiply the spectrums; // the function handles packed spectrum representations well mulSpectrums(tempA, tempB, tempA); // transform the product back from the frequency domain. // Even though all the result rows will be non-zero, // you need only the first C.rows of them, and thus you // pass nonzeroRows == C.rows dft(tempA, tempA, DFT\_INVERSE + DFT\_SCALE, C.rows); // now copy the result back to C. tempA(Rect(0, 0, C.cols, C.rows)).copyTo(C); // all the temporary buffers will be deallocated automatically } To optimize this sample, consider the following approaches:

* + Since nonzeroRows != 0 is passed to the forward transform calls and since A and B are copied to the top-left corners of tempA and tempB, respectively, it is not necessary to clear the whole tempA and tempB. It is only necessary to clear the tempA.cols - A.cols ( tempB.cols - B.cols) rightmost columns of the matrices.
  + This DFT-based convolution does not have to be applied to the whole big arrays, especially if B is significantly smaller than A or vice versa. Instead, you can calculate convolution by parts. To do this, you need to split the output array C into multiple tiles. For each tile, estimate which parts of A and B are required to calculate convolution in this tile. If the tiles in C are too small, the speed will decrease a lot because of repeated work. In the ultimate case, when each tile in C is a single pixel, the algorithm becomes equivalent to the naive convolution algorithm. If the tiles are too big, the temporary arrays tempA and tempB become too big and there is also a slowdown because of bad cache locality. So, there is an optimal tile size somewhere in the middle.
  + If different tiles in C can be calculated in parallel and, thus, the convolution is done by parts, the loop can be threaded.

All of the above improvements have been implemented in #matchTemplate and #filter2D . Therefore, by using them, you can get the performance even better than with the above theoretically optimal implementation. Though, those two functions actually calculate cross-correlation, not convolution, so you need to "flip" the second convolution operand B vertically and horizontally using flip . **Note:**

* + An example using the discrete fourier transform can be found at opencv\_source\_code/samples/cpp/dft.cpp
  + (Python) An example using the dft functionality to perform Wiener deconvolution can be found at opencv\_source/samples/python/deconvolution.py
  + (Python) An example rearranging the quadrants of a Fourier image can be found at opencv\_source/samples/python/dft.py

Parameters:src - input array that could be real or complex.dst - output array whose size and type depends on the flags .flags - transformation flags, representing a combination of the #DftFlags nonzeroRows rows of the input array (#DFT\_INVERSE is not set) or only the first nonzeroRows of the output array (#DFT\_INVERSE is set) contain non-zeros, thus, the function can handle the rest of the rows more efficiently and save some time; this technique is very useful for calculating array cross-correlation or convolution using DFT. SEE: dct , getOptimalDFTSize , mulSpectrums, filter2D , matchTemplate , flip , cartToPolar , magnitude , phase

#### dft public static void dft([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int flags, int nonzeroRows)

Performs a forward or inverse Discrete Fourier transform of a 1D or 2D floating-point array. The function cv::dft performs one of the following:

* + Forward the Fourier transform of a 1D vector of N elements: \(Y = F^{(N)} \cdot X,\) where \(F^{(N)}\_{jk}=\exp(-2\pi i j k/N)\) and \(i=\sqrt{-1}\)
  + Inverse the Fourier transform of a 1D vector of N elements: \(\begin{array}{l} X'= \left (F^{(N)} \right )^{-1} \cdot Y = \left (F^{(N)} \right )^\* \cdot y \\ X = (1/N) \cdot X, \end{array}\) where \(F^\*=\left(\textrm{Re}(F^{(N)})-\textrm{Im}(F^{(N)})\right)^T\)
  + Forward the 2D Fourier transform of a M x N matrix: \(Y = F^{(M)} \cdot X \cdot F^{(N)}\)
  + Inverse the 2D Fourier transform of a M x N matrix: \(\begin{array}{l} X'= \left (F^{(M)} \right )^\* \cdot Y \cdot \left (F^{(N)} \right )^\* \\ X = \frac{1}{M \cdot N} \cdot X' \end{array}\)

In case of real (single-channel) data, the output spectrum of the forward Fourier transform or input spectrum of the inverse Fourier transform can be represented in a packed format called \*CCS\* (complex-conjugate-symmetrical). It was borrowed from IPL (Intel\\* Image Processing Library). Here is how 2D \*CCS\* spectrum looks: \(\begin{bmatrix} Re Y\_{0,0} & Re Y\_{0,1} & Im Y\_{0,1} & Re Y\_{0,2} & Im Y\_{0,2} & \cdots & Re Y\_{0,N/2-1} & Im Y\_{0,N/2-1} & Re Y\_{0,N/2} \\ Re Y\_{1,0} & Re Y\_{1,1} & Im Y\_{1,1} & Re Y\_{1,2} & Im Y\_{1,2} & \cdots & Re Y\_{1,N/2-1} & Im Y\_{1,N/2-1} & Re Y\_{1,N/2} \\ Im Y\_{1,0} & Re Y\_{2,1} & Im Y\_{2,1} & Re Y\_{2,2} & Im Y\_{2,2} & \cdots & Re Y\_{2,N/2-1} & Im Y\_{2,N/2-1} & Im Y\_{1,N/2} \\ \hdotsfor{9} \\ Re Y\_{M/2-1,0} & Re Y\_{M-3,1} & Im Y\_{M-3,1} & \hdotsfor{3} & Re Y\_{M-3,N/2-1} & Im Y\_{M-3,N/2-1}& Re Y\_{M/2-1,N/2} \\ Im Y\_{M/2-1,0} & Re Y\_{M-2,1} & Im Y\_{M-2,1} & \hdotsfor{3} & Re Y\_{M-2,N/2-1} & Im Y\_{M-2,N/2-1}& Im Y\_{M/2-1,N/2} \\ Re Y\_{M/2,0} & Re Y\_{M-1,1} & Im Y\_{M-1,1} & \hdotsfor{3} & Re Y\_{M-1,N/2-1} & Im Y\_{M-1,N/2-1}& Re Y\_{M/2,N/2} \end{bmatrix}\) In case of 1D transform of a real vector, the output looks like the first row of the matrix above. So, the function chooses an operation mode depending on the flags and size of the input array:

* + If #DFT\_ROWS is set or the input array has a single row or single column, the function performs a 1D forward or inverse transform of each row of a matrix when #DFT\_ROWS is set. Otherwise, it performs a 2D transform.
  + If the input array is real and #DFT\_INVERSE is not set, the function performs a forward 1D or 2D transform:
    - When #DFT\_COMPLEX\_OUTPUT is set, the output is a complex matrix of the same size as input.
    - When #DFT\_COMPLEX\_OUTPUT is not set, the output is a real matrix of the same size as input. In case of 2D transform, it uses the packed format as shown above. In case of a single 1D transform, it looks like the first row of the matrix above. In case of multiple 1D transforms (when using the #DFT\_ROWS flag), each row of the output matrix looks like the first row of the matrix above.
  + If the input array is complex and either #DFT\_INVERSE or #DFT\_REAL\_OUTPUT are not set, the output is a complex array of the same size as input. The function performs a forward or inverse 1D or 2D transform of the whole input array or each row of the input array independently, depending on the flags DFT\_INVERSE and DFT\_ROWS.
  + When #DFT\_INVERSE is set and the input array is real, or it is complex but #DFT\_REAL\_OUTPUT is set, the output is a real array of the same size as input. The function performs a 1D or 2D inverse transformation of the whole input array or each individual row, depending on the flags #DFT\_INVERSE and #DFT\_ROWS.

If #DFT\_SCALE is set, the scaling is done after the transformation. Unlike dct , the function supports arrays of arbitrary size. But only those arrays are processed efficiently, whose sizes can be factorized in a product of small prime numbers (2, 3, and 5 in the current implementation). Such an efficient DFT size can be calculated using the getOptimalDFTSize method. The sample below illustrates how to calculate a DFT-based convolution of two 2D real arrays: void convolveDFT(InputArray A, InputArray B, OutputArray C) { // reallocate the output array if needed C.create(abs(A.rows - B.rows)+1, abs(A.cols - B.cols)+1, A.type()); Size dftSize; // calculate the size of DFT transform dftSize.width = getOptimalDFTSize(A.cols + B.cols - 1); dftSize.height = getOptimalDFTSize(A.rows + B.rows - 1); // allocate temporary buffers and initialize them with 0's Mat tempA(dftSize, A.type(), Scalar::all(0)); Mat tempB(dftSize, B.type(), Scalar::all(0)); // copy A and B to the top-left corners of tempA and tempB, respectively Mat roiA(tempA, Rect(0,0,A.cols,A.rows)); A.copyTo(roiA); Mat roiB(tempB, Rect(0,0,B.cols,B.rows)); B.copyTo(roiB); // now transform the padded A & B in-place; // use "nonzeroRows" hint for faster processing dft(tempA, tempA, 0, A.rows); dft(tempB, tempB, 0, B.rows); // multiply the spectrums; // the function handles packed spectrum representations well mulSpectrums(tempA, tempB, tempA); // transform the product back from the frequency domain. // Even though all the result rows will be non-zero, // you need only the first C.rows of them, and thus you // pass nonzeroRows == C.rows dft(tempA, tempA, DFT\_INVERSE + DFT\_SCALE, C.rows); // now copy the result back to C. tempA(Rect(0, 0, C.cols, C.rows)).copyTo(C); // all the temporary buffers will be deallocated automatically } To optimize this sample, consider the following approaches:

* + Since nonzeroRows != 0 is passed to the forward transform calls and since A and B are copied to the top-left corners of tempA and tempB, respectively, it is not necessary to clear the whole tempA and tempB. It is only necessary to clear the tempA.cols - A.cols ( tempB.cols - B.cols) rightmost columns of the matrices.
  + This DFT-based convolution does not have to be applied to the whole big arrays, especially if B is significantly smaller than A or vice versa. Instead, you can calculate convolution by parts. To do this, you need to split the output array C into multiple tiles. For each tile, estimate which parts of A and B are required to calculate convolution in this tile. If the tiles in C are too small, the speed will decrease a lot because of repeated work. In the ultimate case, when each tile in C is a single pixel, the algorithm becomes equivalent to the naive convolution algorithm. If the tiles are too big, the temporary arrays tempA and tempB become too big and there is also a slowdown because of bad cache locality. So, there is an optimal tile size somewhere in the middle.
  + If different tiles in C can be calculated in parallel and, thus, the convolution is done by parts, the loop can be threaded.

All of the above improvements have been implemented in #matchTemplate and #filter2D . Therefore, by using them, you can get the performance even better than with the above theoretically optimal implementation. Though, those two functions actually calculate cross-correlation, not convolution, so you need to "flip" the second convolution operand B vertically and horizontally using flip . **Note:**

* + An example using the discrete fourier transform can be found at opencv\_source\_code/samples/cpp/dft.cpp
  + (Python) An example using the dft functionality to perform Wiener deconvolution can be found at opencv\_source/samples/python/deconvolution.py
  + (Python) An example rearranging the quadrants of a Fourier image can be found at opencv\_source/samples/python/dft.py

Parameters:src - input array that could be real or complex.dst - output array whose size and type depends on the flags .flags - transformation flags, representing a combination of the #DftFlagsnonzeroRows - when the parameter is not zero, the function assumes that only the first nonzeroRows rows of the input array (#DFT\_INVERSE is not set) or only the first nonzeroRows of the output array (#DFT\_INVERSE is set) contain non-zeros, thus, the function can handle the rest of the rows more efficiently and save some time; this technique is very useful for calculating array cross-correlation or convolution using DFT. SEE: dct , getOptimalDFTSize , mulSpectrums, filter2D , matchTemplate , flip , cartToPolar , magnitude , phase

#### divide public static void divide(double scale, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst)

#### divide public static void divide(double scale, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int dtype)

#### divide public static void divide([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Performs per-element division of two arrays or a scalar by an array. The function cv::divide divides one array by another: \(\texttt{dst(I) = saturate(src1(I)\*scale/src2(I))}\) or a scalar by an array when there is no src1 : \(\texttt{dst(I) = saturate(scale/src2(I))}\) When src2(I) is zero, dst(I) will also be zero. Different channels of multi-channel arrays are processed independently. **Note:** Saturation is not applied when the output array has the depth CV\_32S. You may even get result of an incorrect sign in the case of overflow.Parameters:src1 - first input array.src2 - second input array of the same size and type as src1.dst - output array of the same size and type as src2. case of an array-by-array division, you can only pass -1 when src1.depth()==src2.depth(). SEE: multiply, add, subtract

#### divide public static void divide([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double scale) Performs per-element division of two arrays or a scalar by an array. The function cv::divide divides one array by another: \(\texttt{dst(I) = saturate(src1(I)\*scale/src2(I))}\) or a scalar by an array when there is no src1 : \(\texttt{dst(I) = saturate(scale/src2(I))}\) When src2(I) is zero, dst(I) will also be zero. Different channels of multi-channel arrays are processed independently. **Note:** Saturation is not applied when the output array has the depth CV\_32S. You may even get result of an incorrect sign in the case of overflow.Parameters:src1 - first input array.src2 - second input array of the same size and type as src1.scale - scalar factor.dst - output array of the same size and type as src2. case of an array-by-array division, you can only pass -1 when src1.depth()==src2.depth(). SEE: multiply, add, subtract

#### divide public static void divide([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double scale, int dtype) Performs per-element division of two arrays or a scalar by an array. The function cv::divide divides one array by another: \(\texttt{dst(I) = saturate(src1(I)\*scale/src2(I))}\) or a scalar by an array when there is no src1 : \(\texttt{dst(I) = saturate(scale/src2(I))}\) When src2(I) is zero, dst(I) will also be zero. Different channels of multi-channel arrays are processed independently. **Note:** Saturation is not applied when the output array has the depth CV\_32S. You may even get result of an incorrect sign in the case of overflow.Parameters:src1 - first input array.src2 - second input array of the same size and type as src1.scale - scalar factor.dst - output array of the same size and type as src2.dtype - optional depth of the output array; if -1, dst will have depth src2.depth(), but in case of an array-by-array division, you can only pass -1 when src1.depth()==src2.depth(). SEE: multiply, add, subtract

#### divide public static void divide([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst)

#### divide public static void divide([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double scale)

#### divide public static void divide([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double scale, int dtype)

#### eigen public static boolean eigen([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) eigenvalues) Calculates eigenvalues and eigenvectors of a symmetric matrix. The function cv::eigen calculates just eigenvalues, or eigenvalues and eigenvectors of the symmetric matrix src: src\*eigenvectors.row(i).t() = eigenvalues.at<srcType>(i)\*eigenvectors.row(i).t() **Note:** Use cv::eigenNonSymmetric for calculation of real eigenvalues and eigenvectors of non-symmetric matrix.Parameters:src - input matrix that must have CV\_32FC1 or CV\_64FC1 type, square size and be symmetrical (src ^T^ == src).eigenvalues - output vector of eigenvalues of the same type as src; the eigenvalues are stored in the descending order. eigenvectors are stored as subsequent matrix rows, in the same order as the corresponding eigenvalues. SEE: eigenNonSymmetric, completeSymm , PCA Returns:automatically generated

#### eigen public static boolean eigen([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) eigenvalues, [Mat](http://docs.google.com/org/opencv/core/Mat.html) eigenvectors) Calculates eigenvalues and eigenvectors of a symmetric matrix. The function cv::eigen calculates just eigenvalues, or eigenvalues and eigenvectors of the symmetric matrix src: src\*eigenvectors.row(i).t() = eigenvalues.at<srcType>(i)\*eigenvectors.row(i).t() **Note:** Use cv::eigenNonSymmetric for calculation of real eigenvalues and eigenvectors of non-symmetric matrix.Parameters:src - input matrix that must have CV\_32FC1 or CV\_64FC1 type, square size and be symmetrical (src ^T^ == src).eigenvalues - output vector of eigenvalues of the same type as src; the eigenvalues are stored in the descending order.eigenvectors - output matrix of eigenvectors; it has the same size and type as src; the eigenvectors are stored as subsequent matrix rows, in the same order as the corresponding eigenvalues. SEE: eigenNonSymmetric, completeSymm , PCA Returns:automatically generated

#### eigenNonSymmetric public static void eigenNonSymmetric([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) eigenvalues, [Mat](http://docs.google.com/org/opencv/core/Mat.html) eigenvectors) Calculates eigenvalues and eigenvectors of a non-symmetric matrix (real eigenvalues only). **Note:** Assumes real eigenvalues. The function calculates eigenvalues and eigenvectors (optional) of the square matrix src: src\*eigenvectors.row(i).t() = eigenvalues.at<srcType>(i)\*eigenvectors.row(i).t()Parameters:src - input matrix (CV\_32FC1 or CV\_64FC1 type).eigenvalues - output vector of eigenvalues (type is the same type as src).eigenvectors - output matrix of eigenvectors (type is the same type as src). The eigenvectors are stored as subsequent matrix rows, in the same order as the corresponding eigenvalues. SEE: eigen

#### exp public static void exp([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Calculates the exponent of every array element. The function cv::exp calculates the exponent of every element of the input array: \(\texttt{dst} [I] = e^{ src(I) }\) The maximum relative error is about 7e-6 for single-precision input and less than 1e-10 for double-precision input. Currently, the function converts denormalized values to zeros on output. Special values (NaN, Inf) are not handled.Parameters:src - input array.dst - output array of the same size and type as src. SEE: log , cartToPolar , polarToCart , phase , pow , sqrt , magnitude

#### extractChannel public static void extractChannel([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int coi) Extracts a single channel from src (coi is 0-based index)Parameters:src - input arraydst - output arraycoi - index of channel to extract SEE: mixChannels, split

#### fastAtan2 public static float fastAtan2(float y, float x) Calculates the angle of a 2D vector in degrees. The function fastAtan2 calculates the full-range angle of an input 2D vector. The angle is measured in degrees and varies from 0 to 360 degrees. The accuracy is about 0.3 degrees.Parameters:x - x-coordinate of the vector.y - y-coordinate of the vector. Returns:automatically generated

#### findFile public static java.lang.String findFile(java.lang.String relative\_path) Try to find requested data file Search directories: 1. Directories passed via addSamplesDataSearchPath() 2. OPENCV\_SAMPLES\_DATA\_PATH\_HINT environment variable 3. OPENCV\_SAMPLES\_DATA\_PATH environment variable If parameter value is not empty and nothing is found then stop searching. 4. Detects build/install path based on: a. current working directory (CWD) b. and/or binary module location (opencv\_core/opencv\_world, doesn't work with static linkage) 5. Scan <source>/{,data,samples/data} directories if build directory is detected or the current directory is in source tree. 6. Scan <install>/share/OpenCV directory if install directory is detected. SEE: cv::utils::findDataFileParameters:relative\_path - Relative path to data file If true, function prints information message and raises cv::Exception. If false, function returns empty result Returns:Returns path (absolute or relative to the current directory) or empty string if file is not found

#### findFile public static java.lang.String findFile(java.lang.String relative\_path, boolean required) Try to find requested data file Search directories: 1. Directories passed via addSamplesDataSearchPath() 2. OPENCV\_SAMPLES\_DATA\_PATH\_HINT environment variable 3. OPENCV\_SAMPLES\_DATA\_PATH environment variable If parameter value is not empty and nothing is found then stop searching. 4. Detects build/install path based on: a. current working directory (CWD) b. and/or binary module location (opencv\_core/opencv\_world, doesn't work with static linkage) 5. Scan <source>/{,data,samples/data} directories if build directory is detected or the current directory is in source tree. 6. Scan <install>/share/OpenCV directory if install directory is detected. SEE: cv::utils::findDataFileParameters:relative\_path - Relative path to data filerequired - Specify "file not found" handling. If true, function prints information message and raises cv::Exception. If false, function returns empty result Returns:Returns path (absolute or relative to the current directory) or empty string if file is not found

#### findFile public static java.lang.String findFile(java.lang.String relative\_path, boolean required, boolean silentMode) Try to find requested data file Search directories: 1. Directories passed via addSamplesDataSearchPath() 2. OPENCV\_SAMPLES\_DATA\_PATH\_HINT environment variable 3. OPENCV\_SAMPLES\_DATA\_PATH environment variable If parameter value is not empty and nothing is found then stop searching. 4. Detects build/install path based on: a. current working directory (CWD) b. and/or binary module location (opencv\_core/opencv\_world, doesn't work with static linkage) 5. Scan <source>/{,data,samples/data} directories if build directory is detected or the current directory is in source tree. 6. Scan <install>/share/OpenCV directory if install directory is detected. SEE: cv::utils::findDataFileParameters:relative\_path - Relative path to data filerequired - Specify "file not found" handling. If true, function prints information message and raises cv::Exception. If false, function returns empty resultsilentMode - Disables messages Returns:Returns path (absolute or relative to the current directory) or empty string if file is not found

#### findFileOrKeep public static java.lang.String findFileOrKeep(java.lang.String relative\_path)

#### findFileOrKeep public static java.lang.String findFileOrKeep(java.lang.String relative\_path, boolean silentMode)

#### findNonZero public static void findNonZero([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) idx) Returns the list of locations of non-zero pixels Given a binary matrix (likely returned from an operation such as threshold(), compare(), >, ==, etc, return all of the non-zero indices as a cv::Mat or std::vector<cv::Point> (x,y) For example: cv::Mat binaryImage; // input, binary image cv::Mat locations; // output, locations of non-zero pixels cv::findNonZero(binaryImage, locations); // access pixel coordinates Point pnt = locations.at<Point>(i); or cv::Mat binaryImage; // input, binary image vector<Point> locations; // output, locations of non-zero pixels cv::findNonZero(binaryImage, locations); // access pixel coordinates Point pnt = locations[i];Parameters:src - single-channel array (type CV\_8UC1)idx - the output array, type of cv::Mat or std::vector<Point>, corresponding to non-zero indices in the input

#### flip public static void flip([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int flipCode) Flips a 2D array around vertical, horizontal, or both axes. The function cv::flip flips the array in one of three different ways (row and column indices are 0-based): \(\texttt{dst} \_{ij} = \left\{ \begin{array}{l l} \texttt{src} \_{\texttt{src.rows}-i-1,j} & if\; \texttt{flipCode} = 0 \\ \texttt{src} \_{i, \texttt{src.cols} -j-1} & if\; \texttt{flipCode} > 0 \\ \texttt{src} \_{ \texttt{src.rows} -i-1, \texttt{src.cols} -j-1} & if\; \texttt{flipCode} < 0 \\ \end{array} \right.\) The example scenarios of using the function are the following: Vertical flipping of the image (flipCode == 0) to switch between top-left and bottom-left image origin. This is a typical operation in video processing on Microsoft Windows\\* OS. Horizontal flipping of the image with the subsequent horizontal shift and absolute difference calculation to check for a vertical-axis symmetry (flipCode > 0). Simultaneous horizontal and vertical flipping of the image with the subsequent shift and absolute difference calculation to check for a central symmetry (flipCode < 0). Reversing the order of point arrays (flipCode > 0 or flipCode == 0).Parameters:src - input array.dst - output array of the same size and type as src.flipCode - a flag to specify how to flip the array; 0 means flipping around the x-axis and positive value (for example, 1) means flipping around y-axis. Negative value (for example, -1) means flipping around both axes. SEE: transpose , repeat , completeSymm

#### gemm public static void gemm([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, double alpha, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src3, double beta, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Performs generalized matrix multiplication. The function cv::gemm performs generalized matrix multiplication similar to the gemm functions in BLAS level 3. For example, gemm(src1, src2, alpha, src3, beta, dst, GEMM\_1\_T + GEMM\_3\_T) corresponds to \(\texttt{dst} = \texttt{alpha} \cdot \texttt{src1} ^T \cdot \texttt{src2} + \texttt{beta} \cdot \texttt{src3} ^T\) In case of complex (two-channel) data, performed a complex matrix multiplication. The function can be replaced with a matrix expression. For example, the above call can be replaced with: dst = alpha\*src1.t()\*src2 + beta\*src3.t();Parameters:src1 - first multiplied input matrix that could be real(CV\_32FC1, CV\_64FC1) or complex(CV\_32FC2, CV\_64FC2).src2 - second multiplied input matrix of the same type as src1.alpha - weight of the matrix product.src3 - third optional delta matrix added to the matrix product; it should have the same type as src1 and src2.beta - weight of src3.dst - output matrix; it has the proper size and the same type as input matrices. SEE: mulTransposed , transform

#### gemm public static void gemm([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, double alpha, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src3, double beta, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int flags) Performs generalized matrix multiplication. The function cv::gemm performs generalized matrix multiplication similar to the gemm functions in BLAS level 3. For example, gemm(src1, src2, alpha, src3, beta, dst, GEMM\_1\_T + GEMM\_3\_T) corresponds to \(\texttt{dst} = \texttt{alpha} \cdot \texttt{src1} ^T \cdot \texttt{src2} + \texttt{beta} \cdot \texttt{src3} ^T\) In case of complex (two-channel) data, performed a complex matrix multiplication. The function can be replaced with a matrix expression. For example, the above call can be replaced with: dst = alpha\*src1.t()\*src2 + beta\*src3.t();Parameters:src1 - first multiplied input matrix that could be real(CV\_32FC1, CV\_64FC1) or complex(CV\_32FC2, CV\_64FC2).src2 - second multiplied input matrix of the same type as src1.alpha - weight of the matrix product.src3 - third optional delta matrix added to the matrix product; it should have the same type as src1 and src2.beta - weight of src3.dst - output matrix; it has the proper size and the same type as input matrices.flags - operation flags (cv::GemmFlags) SEE: mulTransposed , transform

#### getBuildInformation public static java.lang.String getBuildInformation() Returns full configuration time cmake output. Returned value is raw cmake output including version control system revision, compiler version, compiler flags, enabled modules and third party libraries, etc. Output format depends on target architecture.Returns:automatically generated

#### getCPUTickCount public static long getCPUTickCount() Returns the number of CPU ticks. The function returns the current number of CPU ticks on some architectures (such as x86, x64, PowerPC). On other platforms the function is equivalent to getTickCount. It can also be used for very accurate time measurements, as well as for RNG initialization. Note that in case of multi-CPU systems a thread, from which getCPUTickCount is called, can be suspended and resumed at another CPU with its own counter. So, theoretically (and practically) the subsequent calls to the function do not necessary return the monotonously increasing values. Also, since a modern CPU varies the CPU frequency depending on the load, the number of CPU clocks spent in some code cannot be directly converted to time units. Therefore, getTickCount is generally a preferable solution for measuring execution time.Returns:automatically generated

#### getHardwareFeatureName public static java.lang.String getHardwareFeatureName(int feature) Returns feature name by ID Returns empty string if feature is not definedParameters:feature - automatically generated Returns:automatically generated

#### getIppVersion public static java.lang.String getIppVersion()

#### getNumberOfCPUs public static int getNumberOfCPUs() Returns the number of logical CPUs available for the process.Returns:automatically generated

#### getNumThreads public static int getNumThreads()

Returns the number of threads used by OpenCV for parallel regions. Always returns 1 if OpenCV is built without threading support. The exact meaning of return value depends on the threading framework used by OpenCV library:

* + TBB - The number of threads, that OpenCV will try to use for parallel regions. If there is any tbb::thread\_scheduler\_init in user code conflicting with OpenCV, then function returns default number of threads used by TBB library.
  + OpenMP - An upper bound on the number of threads that could be used to form a new team.
  + Concurrency - The number of threads, that OpenCV will try to use for parallel regions.
  + GCD - Unsupported; returns the GCD thread pool limit (512) for compatibility.
  + C= - The number of threads, that OpenCV will try to use for parallel regions, if before called setNumThreads with threads > 0, otherwise returns the number of logical CPUs, available for the process. SEE: setNumThreads, getThreadNum

Returns:automatically generated

#### getOptimalDFTSize public static int getOptimalDFTSize(int vecsize) Returns the optimal DFT size for a given vector size. DFT performance is not a monotonic function of a vector size. Therefore, when you calculate convolution of two arrays or perform the spectral analysis of an array, it usually makes sense to pad the input data with zeros to get a bit larger array that can be transformed much faster than the original one. Arrays whose size is a power-of-two (2, 4, 8, 16, 32, ...) are the fastest to process. Though, the arrays whose size is a product of 2's, 3's, and 5's (for example, 300 = 5\\*5\\*3\\*2\\*2) are also processed quite efficiently. The function cv::getOptimalDFTSize returns the minimum number N that is greater than or equal to vecsize so that the DFT of a vector of size N can be processed efficiently. In the current implementation N = 2 ^p^ \\* 3 ^q^ \\* 5 ^r^ for some integer p, q, r. The function returns a negative number if vecsize is too large (very close to INT\_MAX ). While the function cannot be used directly to estimate the optimal vector size for DCT transform (since the current DCT implementation supports only even-size vectors), it can be easily processed as getOptimalDFTSize((vecsize+1)/2)\\*2.Parameters:vecsize - vector size. SEE: dft , dct , idft , idct , mulSpectrums Returns:automatically generated

#### getTickCount public static long getTickCount() Returns the number of ticks. The function returns the number of ticks after the certain event (for example, when the machine was turned on). It can be used to initialize RNG or to measure a function execution time by reading the tick count before and after the function call. SEE: getTickFrequency, TickMeterReturns:automatically generated

#### getTickFrequency public static double getTickFrequency() Returns the number of ticks per second. The function returns the number of ticks per second. That is, the following code computes the execution time in seconds: double t = (double)getTickCount(); // do something ... t = ((double)getTickCount() - t)/getTickFrequency(); SEE: getTickCount, TickMeterReturns:automatically generated

#### getVersionMajor public static int getVersionMajor() Returns major library versionReturns:automatically generated

#### getVersionMinor public static int getVersionMinor() Returns minor library versionReturns:automatically generated

#### getVersionRevision public static int getVersionRevision() Returns revision field of the library versionReturns:automatically generated

#### getVersionString public static java.lang.String getVersionString() Returns library version string For example "3.4.1-dev". SEE: getMajorVersion, getMinorVersion, getRevisionVersionReturns:automatically generated

#### hconcat public static void hconcat(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) std::vector<cv::Mat> matrices = { cv::Mat(4, 1, CV\_8UC1, cv::Scalar(1)), cv::Mat(4, 1, CV\_8UC1, cv::Scalar(2)), cv::Mat(4, 1, CV\_8UC1, cv::Scalar(3)),}; cv::Mat out; cv::hconcat( matrices, out ); //out: //[1, 2, 3; // 1, 2, 3; // 1, 2, 3; // 1, 2, 3]Parameters:src - input array or vector of matrices. all of the matrices must have the same number of rows and the same depth.dst - output array. It has the same number of rows and depth as the src, and the sum of cols of the src. same depth.

#### idct public static void idct([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Calculates the inverse Discrete Cosine Transform of a 1D or 2D array. idct(src, dst, flags) is equivalent to dct(src, dst, flags | DCT\_INVERSE).Parameters:src - input floating-point single-channel array.dst - output array of the same size and type as src. SEE: dct, dft, idft, getOptimalDFTSize

#### idct public static void idct([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int flags) Calculates the inverse Discrete Cosine Transform of a 1D or 2D array. idct(src, dst, flags) is equivalent to dct(src, dst, flags | DCT\_INVERSE).Parameters:src - input floating-point single-channel array.dst - output array of the same size and type as src.flags - operation flags. SEE: dct, dft, idft, getOptimalDFTSize

#### idft public static void idft([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Calculates the inverse Discrete Fourier Transform of a 1D or 2D array. idft(src, dst, flags) is equivalent to dft(src, dst, flags | #DFT\_INVERSE) . **Note:** None of dft and idft scales the result by default. So, you should pass #DFT\_SCALE to one of dft or idft explicitly to make these transforms mutually inverse. SEE: dft, dct, idct, mulSpectrums, getOptimalDFTSizeParameters:src - input floating-point real or complex array.dst - output array whose size and type depend on the flags. the convolution sample in dft description.

#### idft public static void idft([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int flags) Calculates the inverse Discrete Fourier Transform of a 1D or 2D array. idft(src, dst, flags) is equivalent to dft(src, dst, flags | #DFT\_INVERSE) . **Note:** None of dft and idft scales the result by default. So, you should pass #DFT\_SCALE to one of dft or idft explicitly to make these transforms mutually inverse. SEE: dft, dct, idct, mulSpectrums, getOptimalDFTSizeParameters:src - input floating-point real or complex array.dst - output array whose size and type depend on the flags.flags - operation flags (see dft and #DftFlags). the convolution sample in dft description.

#### idft public static void idft([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int flags, int nonzeroRows) Calculates the inverse Discrete Fourier Transform of a 1D or 2D array. idft(src, dst, flags) is equivalent to dft(src, dst, flags | #DFT\_INVERSE) . **Note:** None of dft and idft scales the result by default. So, you should pass #DFT\_SCALE to one of dft or idft explicitly to make these transforms mutually inverse. SEE: dft, dct, idct, mulSpectrums, getOptimalDFTSizeParameters:src - input floating-point real or complex array.dst - output array whose size and type depend on the flags.flags - operation flags (see dft and #DftFlags).nonzeroRows - number of dst rows to process; the rest of the rows have undefined content (see the convolution sample in dft description.

#### inRange public static void inRange([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) lowerb, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) upperb, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst)

Checks if array elements lie between the elements of two other arrays. The function checks the range as follows:

* + For every element of a single-channel input array: \(\texttt{dst} (I)= \texttt{lowerb} (I)\_0 \leq \texttt{src} (I)\_0 \leq \texttt{upperb} (I)\_0\)
  + For two-channel arrays: \(\texttt{dst} (I)= \texttt{lowerb} (I)\_0 \leq \texttt{src} (I)\_0 \leq \texttt{upperb} (I)\_0 \land \texttt{lowerb} (I)\_1 \leq \texttt{src} (I)\_1 \leq \texttt{upperb} (I)\_1\)
  + and so forth.

That is, dst (I) is set to 255 (all 1 -bits) if src (I) is within the specified 1D, 2D, 3D, ... box and 0 otherwise. When the lower and/or upper boundary parameters are scalars, the indexes (I) at lowerb and upperb in the above formulas should be omitted.Parameters:src - first input array.lowerb - inclusive lower boundary array or a scalar.upperb - inclusive upper boundary array or a scalar.dst - output array of the same size as src and CV\_8U type.

#### insertChannel public static void insertChannel([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int coi) Inserts a single channel to dst (coi is 0-based index)Parameters:src - input arraydst - output arraycoi - index of channel for insertion SEE: mixChannels, merge

#### invert public static double invert([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Finds the inverse or pseudo-inverse of a matrix. The function cv::invert inverts the matrix src and stores the result in dst . When the matrix src is singular or non-square, the function calculates the pseudo-inverse matrix (the dst matrix) so that norm(src\\*dst - I) is minimal, where I is an identity matrix. In case of the #DECOMP\_LU method, the function returns non-zero value if the inverse has been successfully calculated and 0 if src is singular. In case of the #DECOMP\_SVD method, the function returns the inverse condition number of src (the ratio of the smallest singular value to the largest singular value) and 0 if src is singular. The SVD method calculates a pseudo-inverse matrix if src is singular. Similarly to #DECOMP\_LU, the method #DECOMP\_CHOLESKY works only with non-singular square matrices that should also be symmetrical and positively defined. In this case, the function stores the inverted matrix in dst and returns non-zero. Otherwise, it returns 0.Parameters:src - input floating-point M x N matrix.dst - output matrix of N x M size and the same type as src. SEE: solve, SVD Returns:automatically generated

#### invert public static double invert([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int flags) Finds the inverse or pseudo-inverse of a matrix. The function cv::invert inverts the matrix src and stores the result in dst . When the matrix src is singular or non-square, the function calculates the pseudo-inverse matrix (the dst matrix) so that norm(src\\*dst - I) is minimal, where I is an identity matrix. In case of the #DECOMP\_LU method, the function returns non-zero value if the inverse has been successfully calculated and 0 if src is singular. In case of the #DECOMP\_SVD method, the function returns the inverse condition number of src (the ratio of the smallest singular value to the largest singular value) and 0 if src is singular. The SVD method calculates a pseudo-inverse matrix if src is singular. Similarly to #DECOMP\_LU, the method #DECOMP\_CHOLESKY works only with non-singular square matrices that should also be symmetrical and positively defined. In this case, the function stores the inverted matrix in dst and returns non-zero. Otherwise, it returns 0.Parameters:src - input floating-point M x N matrix.dst - output matrix of N x M size and the same type as src.flags - inversion method (cv::DecompTypes) SEE: solve, SVD Returns:automatically generated

#### kmeans public static double kmeans([Mat](http://docs.google.com/org/opencv/core/Mat.html) data, int K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) bestLabels, [TermCriteria](http://docs.google.com/org/opencv/core/TermCriteria.html) criteria, int attempts, int flags)

Finds centers of clusters and groups input samples around the clusters. The function kmeans implements a k-means algorithm that finds the centers of cluster\_count clusters and groups the input samples around the clusters. As an output, \(\texttt{bestLabels}\_i\) contains a 0-based cluster index for the sample stored in the \(i^{th}\) row of the samples matrix. **Note:**

* + (Python) An example on K-means clustering can be found at opencv\_source\_code/samples/python/kmeans.py

Parameters:data - Data for clustering. An array of N-Dimensional points with float coordinates is needed. Examples of this array can be:

* Mat points(count, 2, CV\_32F);
* Mat points(count, 1, CV\_32FC2);
* Mat points(1, count, CV\_32FC2);
* std::vector<cv::Point2f> points(sampleCount);K - Number of clusters to split the set by.bestLabels - Input/output integer array that stores the cluster indices for every sample.criteria - The algorithm termination criteria, that is, the maximum number of iterations and/or the desired accuracy. The accuracy is specified as criteria.epsilon. As soon as each of the cluster centers moves by less than criteria.epsilon on some iteration, the algorithm stops.attempts - Flag to specify the number of times the algorithm is executed using different initial labellings. The algorithm returns the labels that yield the best compactness (see the last function parameter).flags - Flag that can take values of cv::KmeansFlags Returns:The function returns the compactness measure that is computed as \(\sum \_i \| \texttt{samples} \_i - \texttt{centers} \_{ \texttt{labels} \_i} \| ^2\) after every attempt. The best (minimum) value is chosen and the corresponding labels and the compactness value are returned by the function. Basically, you can use only the core of the function, set the number of attempts to 1, initialize labels each time using a custom algorithm, pass them with the ( flags = #KMEANS\_USE\_INITIAL\_LABELS ) flag, and then choose the best (most-compact) clustering.

#### kmeans public static double kmeans([Mat](http://docs.google.com/org/opencv/core/Mat.html) data, int K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) bestLabels, [TermCriteria](http://docs.google.com/org/opencv/core/TermCriteria.html) criteria, int attempts, int flags, [Mat](http://docs.google.com/org/opencv/core/Mat.html) centers)

Finds centers of clusters and groups input samples around the clusters. The function kmeans implements a k-means algorithm that finds the centers of cluster\_count clusters and groups the input samples around the clusters. As an output, \(\texttt{bestLabels}\_i\) contains a 0-based cluster index for the sample stored in the \(i^{th}\) row of the samples matrix. **Note:**

* + (Python) An example on K-means clustering can be found at opencv\_source\_code/samples/python/kmeans.py

Parameters:data - Data for clustering. An array of N-Dimensional points with float coordinates is needed. Examples of this array can be:

* Mat points(count, 2, CV\_32F);
* Mat points(count, 1, CV\_32FC2);
* Mat points(1, count, CV\_32FC2);
* std::vector<cv::Point2f> points(sampleCount);K - Number of clusters to split the set by.bestLabels - Input/output integer array that stores the cluster indices for every sample.criteria - The algorithm termination criteria, that is, the maximum number of iterations and/or the desired accuracy. The accuracy is specified as criteria.epsilon. As soon as each of the cluster centers moves by less than criteria.epsilon on some iteration, the algorithm stops.attempts - Flag to specify the number of times the algorithm is executed using different initial labellings. The algorithm returns the labels that yield the best compactness (see the last function parameter).flags - Flag that can take values of cv::KmeansFlagscenters - Output matrix of the cluster centers, one row per each cluster center. Returns:The function returns the compactness measure that is computed as \(\sum \_i \| \texttt{samples} \_i - \texttt{centers} \_{ \texttt{labels} \_i} \| ^2\) after every attempt. The best (minimum) value is chosen and the corresponding labels and the compactness value are returned by the function. Basically, you can use only the core of the function, set the number of attempts to 1, initialize labels each time using a custom algorithm, pass them with the ( flags = #KMEANS\_USE\_INITIAL\_LABELS ) flag, and then choose the best (most-compact) clustering.

#### log public static void log([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Calculates the natural logarithm of every array element. The function cv::log calculates the natural logarithm of every element of the input array: \(\texttt{dst} (I) = \log (\texttt{src}(I)) \) Output on zero, negative and special (NaN, Inf) values is undefined.Parameters:src - input array.dst - output array of the same size and type as src . SEE: exp, cartToPolar, polarToCart, phase, pow, sqrt, magnitude

#### LUT public static void LUT([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) lut, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Performs a look-up table transform of an array. The function LUT fills the output array with values from the look-up table. Indices of the entries are taken from the input array. That is, the function processes each element of src as follows: \(\texttt{dst} (I) \leftarrow \texttt{lut(src(I) + d)}\) where \(d = \fork{0}{if \(\texttt{src}\) has depth \(\texttt{CV\_8U}\)}{128}{if \(\texttt{src}\) has depth \(\texttt{CV\_8S}\)}\)Parameters:src - input array of 8-bit elements.lut - look-up table of 256 elements; in case of multi-channel input array, the table should either have a single channel (in this case the same table is used for all channels) or the same number of channels as in the input array.dst - output array of the same size and number of channels as src, and the same depth as lut. SEE: convertScaleAbs, Mat::convertTo

#### magnitude public static void magnitude([Mat](http://docs.google.com/org/opencv/core/Mat.html) x, [Mat](http://docs.google.com/org/opencv/core/Mat.html) y, [Mat](http://docs.google.com/org/opencv/core/Mat.html) magnitude) Calculates the magnitude of 2D vectors. The function cv::magnitude calculates the magnitude of 2D vectors formed from the corresponding elements of x and y arrays: \(\texttt{dst} (I) = \sqrt{\texttt{x}(I)^2 + \texttt{y}(I)^2}\)Parameters:x - floating-point array of x-coordinates of the vectors.y - floating-point array of y-coordinates of the vectors; it must have the same size as x.magnitude - output array of the same size and type as x. SEE: cartToPolar, polarToCart, phase, sqrt

#### Mahalanobis public static double Mahalanobis([Mat](http://docs.google.com/org/opencv/core/Mat.html) v1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) v2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) icovar) Calculates the Mahalanobis distance between two vectors. The function cv::Mahalanobis calculates and returns the weighted distance between two vectors: \(d( \texttt{vec1} , \texttt{vec2} )= \sqrt{\sum\_{i,j}{\texttt{icovar(i,j)}\cdot(\texttt{vec1}(I)-\texttt{vec2}(I))\cdot(\texttt{vec1(j)}-\texttt{vec2(j)})} }\) The covariance matrix may be calculated using the #calcCovarMatrix function and then inverted using the invert function (preferably using the #DECOMP\_SVD method, as the most accurate).Parameters:v1 - first 1D input vector.v2 - second 1D input vector.icovar - inverse covariance matrix. Returns:automatically generated

#### max public static void max([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Calculates per-element maximum of two arrays or an array and a scalar. The function cv::max calculates the per-element maximum of two arrays: \(\texttt{dst} (I)= \max ( \texttt{src1} (I), \texttt{src2} (I))\) or array and a scalar: \(\texttt{dst} (I)= \max ( \texttt{src1} (I), \texttt{value} )\)Parameters:src1 - first input array.src2 - second input array of the same size and type as src1 .dst - output array of the same size and type as src1. SEE: min, compare, inRange, minMaxLoc, REF: MatrixExpressions

#### max public static void max([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst)

#### mean public static [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) mean([Mat](http://docs.google.com/org/opencv/core/Mat.html) src) Calculates an average (mean) of array elements. The function cv::mean calculates the mean value M of array elements, independently for each channel, and return it: \(\begin{array}{l} N = \sum \_{I: \; \texttt{mask} (I) \ne 0} 1 \\ M\_c = \left ( \sum \_{I: \; \texttt{mask} (I) \ne 0}{ \texttt{mtx} (I)\_c} \right )/N \end{array}\) When all the mask elements are 0's, the function returns Scalar::all(0)Parameters:src - input array that should have from 1 to 4 channels so that the result can be stored in Scalar\_ . SEE: countNonZero, meanStdDev, norm, minMaxLoc Returns:automatically generated

#### mean public static [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) mean([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) Calculates an average (mean) of array elements. The function cv::mean calculates the mean value M of array elements, independently for each channel, and return it: \(\begin{array}{l} N = \sum \_{I: \; \texttt{mask} (I) \ne 0} 1 \\ M\_c = \left ( \sum \_{I: \; \texttt{mask} (I) \ne 0}{ \texttt{mtx} (I)\_c} \right )/N \end{array}\) When all the mask elements are 0's, the function returns Scalar::all(0)Parameters:src - input array that should have from 1 to 4 channels so that the result can be stored in Scalar\_ .mask - optional operation mask. SEE: countNonZero, meanStdDev, norm, minMaxLoc Returns:automatically generated

#### meanStdDev public static void meanStdDev([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [MatOfDouble](http://docs.google.com/org/opencv/core/MatOfDouble.html) mean, [MatOfDouble](http://docs.google.com/org/opencv/core/MatOfDouble.html) stddev) Calculates a mean and standard deviation of array elements. The function cv::meanStdDev calculates the mean and the standard deviation M of array elements independently for each channel and returns it via the output parameters: \(\begin{array}{l} N = \sum \_{I, \texttt{mask} (I) \ne 0} 1 \\ \texttt{mean} \_c = \frac{\sum\_{ I: \; \texttt{mask}(I) \ne 0} \texttt{src} (I)\_c}{N} \\ \texttt{stddev} \_c = \sqrt{\frac{\sum\_{ I: \; \texttt{mask}(I) \ne 0} \left ( \texttt{src} (I)\_c - \texttt{mean} \_c \right )^2}{N}} \end{array}\) When all the mask elements are 0's, the function returns mean=stddev=Scalar::all(0). **Note:** The calculated standard deviation is only the diagonal of the complete normalized covariance matrix. If the full matrix is needed, you can reshape the multi-channel array M x N to the single-channel array M\\*N x mtx.channels() (only possible when the matrix is continuous) and then pass the matrix to calcCovarMatrix .Parameters:src - input array that should have from 1 to 4 channels so that the results can be stored in Scalar\_ 's.mean - output parameter: calculated mean value.stddev - output parameter: calculated standard deviation. SEE: countNonZero, mean, norm, minMaxLoc, calcCovarMatrix

#### meanStdDev public static void meanStdDev([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [MatOfDouble](http://docs.google.com/org/opencv/core/MatOfDouble.html) mean, [MatOfDouble](http://docs.google.com/org/opencv/core/MatOfDouble.html) stddev, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) Calculates a mean and standard deviation of array elements. The function cv::meanStdDev calculates the mean and the standard deviation M of array elements independently for each channel and returns it via the output parameters: \(\begin{array}{l} N = \sum \_{I, \texttt{mask} (I) \ne 0} 1 \\ \texttt{mean} \_c = \frac{\sum\_{ I: \; \texttt{mask}(I) \ne 0} \texttt{src} (I)\_c}{N} \\ \texttt{stddev} \_c = \sqrt{\frac{\sum\_{ I: \; \texttt{mask}(I) \ne 0} \left ( \texttt{src} (I)\_c - \texttt{mean} \_c \right )^2}{N}} \end{array}\) When all the mask elements are 0's, the function returns mean=stddev=Scalar::all(0). **Note:** The calculated standard deviation is only the diagonal of the complete normalized covariance matrix. If the full matrix is needed, you can reshape the multi-channel array M x N to the single-channel array M\\*N x mtx.channels() (only possible when the matrix is continuous) and then pass the matrix to calcCovarMatrix .Parameters:src - input array that should have from 1 to 4 channels so that the results can be stored in Scalar\_ 's.mean - output parameter: calculated mean value.stddev - output parameter: calculated standard deviation.mask - optional operation mask. SEE: countNonZero, mean, norm, minMaxLoc, calcCovarMatrix

#### merge public static void merge(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> mv, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst)Parameters:mv - input vector of matrices to be merged; all the matrices in mv must have the same size and the same depth.dst - output array of the same size and the same depth as mv[0]; The number of channels will be the total number of channels in the matrix array.

#### min public static void min([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Calculates per-element minimum of two arrays or an array and a scalar. The function cv::min calculates the per-element minimum of two arrays: \(\texttt{dst} (I)= \min ( \texttt{src1} (I), \texttt{src2} (I))\) or array and a scalar: \(\texttt{dst} (I)= \min ( \texttt{src1} (I), \texttt{value} )\)Parameters:src1 - first input array.src2 - second input array of the same size and type as src1.dst - output array of the same size and type as src1. SEE: max, compare, inRange, minMaxLoc

#### min public static void min([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst)

#### minMaxLoc public static [Core.MinMaxLocResult](http://docs.google.com/org/opencv/core/Core.MinMaxLocResult.html) minMaxLoc([Mat](http://docs.google.com/org/opencv/core/Mat.html) src)

#### minMaxLoc public static [Core.MinMaxLocResult](http://docs.google.com/org/opencv/core/Core.MinMaxLocResult.html) minMaxLoc([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask)

#### mixChannels public static void mixChannels(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> src, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> dst, [MatOfInt](http://docs.google.com/org/opencv/core/MatOfInt.html) fromTo)Parameters:src - input array or vector of matrices; all of the matrices must have the same size and the same depth.dst - output array or vector of matrices; all the matrices **must be allocated**; their size and depth must be the same as in src[0].fromTo - array of index pairs specifying which channels are copied and where; fromTo[k\\*2] is a 0-based index of the input channel in src, fromTo[k\\*2+1] is an index of the output channel in dst; the continuous channel numbering is used: the first input image channels are indexed from 0 to src[0].channels()-1, the second input image channels are indexed from src[0].channels() to src[0].channels() + src[1].channels()-1, and so on, the same scheme is used for the output image channels; as a special case, when fromTo[k\\*2] is negative, the corresponding output channel is filled with zero .

#### mulSpectrums public static void mulSpectrums([Mat](http://docs.google.com/org/opencv/core/Mat.html) a, [Mat](http://docs.google.com/org/opencv/core/Mat.html) b, [Mat](http://docs.google.com/org/opencv/core/Mat.html) c, int flags) Performs the per-element multiplication of two Fourier spectrums. The function cv::mulSpectrums performs the per-element multiplication of the two CCS-packed or complex matrices that are results of a real or complex Fourier transform. The function, together with dft and idft , may be used to calculate convolution (pass conjB=false ) or correlation (pass conjB=true ) of two arrays rapidly. When the arrays are complex, they are simply multiplied (per element) with an optional conjugation of the second-array elements. When the arrays are real, they are assumed to be CCS-packed (see dft for details).Parameters:a - first input array.b - second input array of the same size and type as src1 .c - output array of the same size and type as src1 .flags - operation flags; currently, the only supported flag is cv::DFT\_ROWS, which indicates that each row of src1 and src2 is an independent 1D Fourier spectrum. If you do not want to use this flag, then simply add a 0 as value. or not (false).

#### mulSpectrums public static void mulSpectrums([Mat](http://docs.google.com/org/opencv/core/Mat.html) a, [Mat](http://docs.google.com/org/opencv/core/Mat.html) b, [Mat](http://docs.google.com/org/opencv/core/Mat.html) c, int flags, boolean conjB) Performs the per-element multiplication of two Fourier spectrums. The function cv::mulSpectrums performs the per-element multiplication of the two CCS-packed or complex matrices that are results of a real or complex Fourier transform. The function, together with dft and idft , may be used to calculate convolution (pass conjB=false ) or correlation (pass conjB=true ) of two arrays rapidly. When the arrays are complex, they are simply multiplied (per element) with an optional conjugation of the second-array elements. When the arrays are real, they are assumed to be CCS-packed (see dft for details).Parameters:a - first input array.b - second input array of the same size and type as src1 .c - output array of the same size and type as src1 .flags - operation flags; currently, the only supported flag is cv::DFT\_ROWS, which indicates that each row of src1 and src2 is an independent 1D Fourier spectrum. If you do not want to use this flag, then simply add a 0 as value.conjB - optional flag that conjugates the second input array before the multiplication (true) or not (false).

#### multiply public static void multiply([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Calculates the per-element scaled product of two arrays. The function multiply calculates the per-element product of two arrays: \(\texttt{dst} (I)= \texttt{saturate} ( \texttt{scale} \cdot \texttt{src1} (I) \cdot \texttt{src2} (I))\) There is also a REF: MatrixExpressions -friendly variant of the first function. See Mat::mul . For a not-per-element matrix product, see gemm . **Note:** Saturation is not applied when the output array has the depth CV\_32S. You may even get result of an incorrect sign in the case of overflow.Parameters:src1 - first input array.src2 - second input array of the same size and the same type as src1.dst - output array of the same size and type as src1. SEE: add, subtract, divide, scaleAdd, addWeighted, accumulate, accumulateProduct, accumulateSquare, Mat::convertTo

#### multiply public static void multiply([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double scale) Calculates the per-element scaled product of two arrays. The function multiply calculates the per-element product of two arrays: \(\texttt{dst} (I)= \texttt{saturate} ( \texttt{scale} \cdot \texttt{src1} (I) \cdot \texttt{src2} (I))\) There is also a REF: MatrixExpressions -friendly variant of the first function. See Mat::mul . For a not-per-element matrix product, see gemm . **Note:** Saturation is not applied when the output array has the depth CV\_32S. You may even get result of an incorrect sign in the case of overflow.Parameters:src1 - first input array.src2 - second input array of the same size and the same type as src1.dst - output array of the same size and type as src1.scale - optional scale factor. SEE: add, subtract, divide, scaleAdd, addWeighted, accumulate, accumulateProduct, accumulateSquare, Mat::convertTo

#### multiply public static void multiply([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double scale, int dtype) Calculates the per-element scaled product of two arrays. The function multiply calculates the per-element product of two arrays: \(\texttt{dst} (I)= \texttt{saturate} ( \texttt{scale} \cdot \texttt{src1} (I) \cdot \texttt{src2} (I))\) There is also a REF: MatrixExpressions -friendly variant of the first function. See Mat::mul . For a not-per-element matrix product, see gemm . **Note:** Saturation is not applied when the output array has the depth CV\_32S. You may even get result of an incorrect sign in the case of overflow.Parameters:src1 - first input array.src2 - second input array of the same size and the same type as src1.dst - output array of the same size and type as src1.scale - optional scale factor.dtype - optional depth of the output array SEE: add, subtract, divide, scaleAdd, addWeighted, accumulate, accumulateProduct, accumulateSquare, Mat::convertTo

#### multiply public static void multiply([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst)

#### multiply public static void multiply([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double scale)

#### multiply public static void multiply([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double scale, int dtype)

#### mulTransposed public static void mulTransposed([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, boolean aTa) Calculates the product of a matrix and its transposition. The function cv::mulTransposed calculates the product of src and its transposition: \(\texttt{dst} = \texttt{scale} ( \texttt{src} - \texttt{delta} )^T ( \texttt{src} - \texttt{delta} )\) if aTa=true , and \(\texttt{dst} = \texttt{scale} ( \texttt{src} - \texttt{delta} ) ( \texttt{src} - \texttt{delta} )^T\) otherwise. The function is used to calculate the covariance matrix. With zero delta, it can be used as a faster substitute for general matrix product A\\*B when B=A'Parameters:src - input single-channel matrix. Note that unlike gemm, the function can multiply not only floating-point matrices.dst - output square matrix.aTa - Flag specifying the multiplication ordering. See the description below. multiplication. When the matrix is empty ( delta=noArray() ), it is assumed to be zero, that is, nothing is subtracted. If it has the same size as src , it is simply subtracted. Otherwise, it is "repeated" (see repeat ) to cover the full src and then subtracted. Type of the delta matrix, when it is not empty, must be the same as the type of created output matrix. See the dtype parameter description below. the output matrix will have the same type as src . Otherwise, it will be type=CV\_MAT\_DEPTH(dtype) that should be either CV\_32F or CV\_64F . SEE: calcCovarMatrix, gemm, repeat, reduce

#### mulTransposed public static void mulTransposed([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, boolean aTa, [Mat](http://docs.google.com/org/opencv/core/Mat.html) delta) Calculates the product of a matrix and its transposition. The function cv::mulTransposed calculates the product of src and its transposition: \(\texttt{dst} = \texttt{scale} ( \texttt{src} - \texttt{delta} )^T ( \texttt{src} - \texttt{delta} )\) if aTa=true , and \(\texttt{dst} = \texttt{scale} ( \texttt{src} - \texttt{delta} ) ( \texttt{src} - \texttt{delta} )^T\) otherwise. The function is used to calculate the covariance matrix. With zero delta, it can be used as a faster substitute for general matrix product A\\*B when B=A'Parameters:src - input single-channel matrix. Note that unlike gemm, the function can multiply not only floating-point matrices.dst - output square matrix.aTa - Flag specifying the multiplication ordering. See the description below.delta - Optional delta matrix subtracted from src before the multiplication. When the matrix is empty ( delta=noArray() ), it is assumed to be zero, that is, nothing is subtracted. If it has the same size as src , it is simply subtracted. Otherwise, it is "repeated" (see repeat ) to cover the full src and then subtracted. Type of the delta matrix, when it is not empty, must be the same as the type of created output matrix. See the dtype parameter description below. the output matrix will have the same type as src . Otherwise, it will be type=CV\_MAT\_DEPTH(dtype) that should be either CV\_32F or CV\_64F . SEE: calcCovarMatrix, gemm, repeat, reduce

#### mulTransposed public static void mulTransposed([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, boolean aTa, [Mat](http://docs.google.com/org/opencv/core/Mat.html) delta, double scale) Calculates the product of a matrix and its transposition. The function cv::mulTransposed calculates the product of src and its transposition: \(\texttt{dst} = \texttt{scale} ( \texttt{src} - \texttt{delta} )^T ( \texttt{src} - \texttt{delta} )\) if aTa=true , and \(\texttt{dst} = \texttt{scale} ( \texttt{src} - \texttt{delta} ) ( \texttt{src} - \texttt{delta} )^T\) otherwise. The function is used to calculate the covariance matrix. With zero delta, it can be used as a faster substitute for general matrix product A\\*B when B=A'Parameters:src - input single-channel matrix. Note that unlike gemm, the function can multiply not only floating-point matrices.dst - output square matrix.aTa - Flag specifying the multiplication ordering. See the description below.delta - Optional delta matrix subtracted from src before the multiplication. When the matrix is empty ( delta=noArray() ), it is assumed to be zero, that is, nothing is subtracted. If it has the same size as src , it is simply subtracted. Otherwise, it is "repeated" (see repeat ) to cover the full src and then subtracted. Type of the delta matrix, when it is not empty, must be the same as the type of created output matrix. See the dtype parameter description below.scale - Optional scale factor for the matrix product. the output matrix will have the same type as src . Otherwise, it will be type=CV\_MAT\_DEPTH(dtype) that should be either CV\_32F or CV\_64F . SEE: calcCovarMatrix, gemm, repeat, reduce

#### mulTransposed public static void mulTransposed([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, boolean aTa, [Mat](http://docs.google.com/org/opencv/core/Mat.html) delta, double scale, int dtype) Calculates the product of a matrix and its transposition. The function cv::mulTransposed calculates the product of src and its transposition: \(\texttt{dst} = \texttt{scale} ( \texttt{src} - \texttt{delta} )^T ( \texttt{src} - \texttt{delta} )\) if aTa=true , and \(\texttt{dst} = \texttt{scale} ( \texttt{src} - \texttt{delta} ) ( \texttt{src} - \texttt{delta} )^T\) otherwise. The function is used to calculate the covariance matrix. With zero delta, it can be used as a faster substitute for general matrix product A\\*B when B=A'Parameters:src - input single-channel matrix. Note that unlike gemm, the function can multiply not only floating-point matrices.dst - output square matrix.aTa - Flag specifying the multiplication ordering. See the description below.delta - Optional delta matrix subtracted from src before the multiplication. When the matrix is empty ( delta=noArray() ), it is assumed to be zero, that is, nothing is subtracted. If it has the same size as src , it is simply subtracted. Otherwise, it is "repeated" (see repeat ) to cover the full src and then subtracted. Type of the delta matrix, when it is not empty, must be the same as the type of created output matrix. See the dtype parameter description below.scale - Optional scale factor for the matrix product.dtype - Optional type of the output matrix. When it is negative, the output matrix will have the same type as src . Otherwise, it will be type=CV\_MAT\_DEPTH(dtype) that should be either CV\_32F or CV\_64F . SEE: calcCovarMatrix, gemm, repeat, reduce

#### norm public static double norm([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1) Calculates the absolute norm of an array. This version of #norm calculates the absolute norm of src1. The type of norm to calculate is specified using #NormTypes. As example for one array consider the function \(r(x)= \begin{pmatrix} x \\ 1-x \end{pmatrix}, x \in [-1;1]\). The \( L\_{1}, L\_{2} \) and \( L\_{\infty} \) norm for the sample value \(r(-1) = \begin{pmatrix} -1 \\ 2 \end{pmatrix}\) is calculated as follows \(align\*} \| r(-1) \|\_{L\_1} &= |-1| + |2| = 3 \\ \| r(-1) \|\_{L\_2} &= \sqrt{(-1)^{2} + (2)^{2}} = \sqrt{5} \\ \| r(-1) \|\_{L\_\infty} &= \max(|-1|,|2|) = 2 \) and for \(r(0.5) = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}\) the calculation is \(align\*} \| r(0.5) \|\_{L\_1} &= |0.5| + |0.5| = 1 \\ \| r(0.5) \|\_{L\_2} &= \sqrt{(0.5)^{2} + (0.5)^{2}} = \sqrt{0.5} \\ \| r(0.5) \|\_{L\_\infty} &= \max(|0.5|,|0.5|) = 0.5. \) The following graphic shows all values for the three norm functions \(\| r(x) \|\_{L\_1}, \| r(x) \|\_{L\_2}\) and \(\| r(x) \|\_{L\_\infty}\). It is notable that the \( L\_{1} \) norm forms the upper and the \( L\_{\infty} \) norm forms the lower border for the example function \( r(x) \). ![Graphs for the different norm functions from the above example](pics/NormTypes\_OneArray\_1-2-INF.png) When the mask parameter is specified and it is not empty, the norm is If normType is not specified, #NORM\_L2 is used. calculated only over the region specified by the mask. Multi-channel input arrays are treated as single-channel arrays, that is, the results for all channels are combined. Hamming norms can only be calculated with CV\_8U depth arrays.Parameters:src1 - first input array. Returns:automatically generated

#### norm public static double norm([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, int normType) Calculates the absolute norm of an array. This version of #norm calculates the absolute norm of src1. The type of norm to calculate is specified using #NormTypes. As example for one array consider the function \(r(x)= \begin{pmatrix} x \\ 1-x \end{pmatrix}, x \in [-1;1]\). The \( L\_{1}, L\_{2} \) and \( L\_{\infty} \) norm for the sample value \(r(-1) = \begin{pmatrix} -1 \\ 2 \end{pmatrix}\) is calculated as follows \(align\*} \| r(-1) \|\_{L\_1} &= |-1| + |2| = 3 \\ \| r(-1) \|\_{L\_2} &= \sqrt{(-1)^{2} + (2)^{2}} = \sqrt{5} \\ \| r(-1) \|\_{L\_\infty} &= \max(|-1|,|2|) = 2 \) and for \(r(0.5) = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}\) the calculation is \(align\*} \| r(0.5) \|\_{L\_1} &= |0.5| + |0.5| = 1 \\ \| r(0.5) \|\_{L\_2} &= \sqrt{(0.5)^{2} + (0.5)^{2}} = \sqrt{0.5} \\ \| r(0.5) \|\_{L\_\infty} &= \max(|0.5|,|0.5|) = 0.5. \) The following graphic shows all values for the three norm functions \(\| r(x) \|\_{L\_1}, \| r(x) \|\_{L\_2}\) and \(\| r(x) \|\_{L\_\infty}\). It is notable that the \( L\_{1} \) norm forms the upper and the \( L\_{\infty} \) norm forms the lower border for the example function \( r(x) \). ![Graphs for the different norm functions from the above example](pics/NormTypes\_OneArray\_1-2-INF.png) When the mask parameter is specified and it is not empty, the norm is If normType is not specified, #NORM\_L2 is used. calculated only over the region specified by the mask. Multi-channel input arrays are treated as single-channel arrays, that is, the results for all channels are combined. Hamming norms can only be calculated with CV\_8U depth arrays.Parameters:src1 - first input array.normType - type of the norm (see #NormTypes). Returns:automatically generated

#### norm public static double norm([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, int normType, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) Calculates the absolute norm of an array. This version of #norm calculates the absolute norm of src1. The type of norm to calculate is specified using #NormTypes. As example for one array consider the function \(r(x)= \begin{pmatrix} x \\ 1-x \end{pmatrix}, x \in [-1;1]\). The \( L\_{1}, L\_{2} \) and \( L\_{\infty} \) norm for the sample value \(r(-1) = \begin{pmatrix} -1 \\ 2 \end{pmatrix}\) is calculated as follows \(align\*} \| r(-1) \|\_{L\_1} &= |-1| + |2| = 3 \\ \| r(-1) \|\_{L\_2} &= \sqrt{(-1)^{2} + (2)^{2}} = \sqrt{5} \\ \| r(-1) \|\_{L\_\infty} &= \max(|-1|,|2|) = 2 \) and for \(r(0.5) = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}\) the calculation is \(align\*} \| r(0.5) \|\_{L\_1} &= |0.5| + |0.5| = 1 \\ \| r(0.5) \|\_{L\_2} &= \sqrt{(0.5)^{2} + (0.5)^{2}} = \sqrt{0.5} \\ \| r(0.5) \|\_{L\_\infty} &= \max(|0.5|,|0.5|) = 0.5. \) The following graphic shows all values for the three norm functions \(\| r(x) \|\_{L\_1}, \| r(x) \|\_{L\_2}\) and \(\| r(x) \|\_{L\_\infty}\). It is notable that the \( L\_{1} \) norm forms the upper and the \( L\_{\infty} \) norm forms the lower border for the example function \( r(x) \). ![Graphs for the different norm functions from the above example](pics/NormTypes\_OneArray\_1-2-INF.png) When the mask parameter is specified and it is not empty, the norm is If normType is not specified, #NORM\_L2 is used. calculated only over the region specified by the mask. Multi-channel input arrays are treated as single-channel arrays, that is, the results for all channels are combined. Hamming norms can only be calculated with CV\_8U depth arrays.Parameters:src1 - first input array.normType - type of the norm (see #NormTypes).mask - optional operation mask; it must have the same size as src1 and CV\_8UC1 type. Returns:automatically generated

#### norm public static double norm([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2) Calculates an absolute difference norm or a relative difference norm. This version of cv::norm calculates the absolute difference norm or the relative difference norm of arrays src1 and src2. The type of norm to calculate is specified using #NormTypes.Parameters:src1 - first input array.src2 - second input array of the same size and the same type as src1. Returns:automatically generated

#### norm public static double norm([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, int normType) Calculates an absolute difference norm or a relative difference norm. This version of cv::norm calculates the absolute difference norm or the relative difference norm of arrays src1 and src2. The type of norm to calculate is specified using #NormTypes.Parameters:src1 - first input array.src2 - second input array of the same size and the same type as src1.normType - type of the norm (see #NormTypes). Returns:automatically generated

#### norm public static double norm([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, int normType, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) Calculates an absolute difference norm or a relative difference norm. This version of cv::norm calculates the absolute difference norm or the relative difference norm of arrays src1 and src2. The type of norm to calculate is specified using #NormTypes.Parameters:src1 - first input array.src2 - second input array of the same size and the same type as src1.normType - type of the norm (see #NormTypes).mask - optional operation mask; it must have the same size as src1 and CV\_8UC1 type. Returns:automatically generated

#### normalize public static void normalize([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Normalizes the norm or value range of an array. The function cv::normalize normalizes scale and shift the input array elements so that \(\| \texttt{dst} \| \_{L\_p}= \texttt{alpha}\) (where p=Inf, 1 or 2) when normType=NORM\_INF, NORM\_L1, or NORM\_L2, respectively; or so that \(\min \_I \texttt{dst} (I)= \texttt{alpha} , \, \, \max \_I \texttt{dst} (I)= \texttt{beta}\) when normType=NORM\_MINMAX (for dense arrays only). The optional mask specifies a sub-array to be normalized. This means that the norm or min-n-max are calculated over the sub-array, and then this sub-array is modified to be normalized. If you want to only use the mask to calculate the norm or min-max but modify the whole array, you can use norm and Mat::convertTo. In case of sparse matrices, only the non-zero values are analyzed and transformed. Because of this, the range transformation for sparse matrices is not allowed since it can shift the zero level. Possible usage with some positive example data: vector<double> positiveData = { 2.0, 8.0, 10.0 }; vector<double> normalizedData\_l1, normalizedData\_l2, normalizedData\_inf, normalizedData\_minmax; // Norm to probability (total count) // sum(numbers) = 20.0 // 2.0 0.1 (2.0/20.0) // 8.0 0.4 (8.0/20.0) // 10.0 0.5 (10.0/20.0) normalize(positiveData, normalizedData\_l1, 1.0, 0.0, NORM\_L1); // Norm to unit vector: ||positiveData|| = 1.0 // 2.0 0.15 // 8.0 0.62 // 10.0 0.77 normalize(positiveData, normalizedData\_l2, 1.0, 0.0, NORM\_L2); // Norm to max element // 2.0 0.2 (2.0/10.0) // 8.0 0.8 (8.0/10.0) // 10.0 1.0 (10.0/10.0) normalize(positiveData, normalizedData\_inf, 1.0, 0.0, NORM\_INF); // Norm to range [0.0;1.0] // 2.0 0.0 (shift to left border) // 8.0 0.75 (6.0/8.0) // 10.0 1.0 (shift to right border) normalize(positiveData, normalizedData\_minmax, 1.0, 0.0, NORM\_MINMAX);Parameters:src - input array.dst - output array of the same size as src . normalization. normalization. number of channels as src and the depth =CV\_MAT\_DEPTH(dtype). SEE: norm, Mat::convertTo, SparseMat::convertTo

#### normalize public static void normalize([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double alpha) Normalizes the norm or value range of an array. The function cv::normalize normalizes scale and shift the input array elements so that \(\| \texttt{dst} \| \_{L\_p}= \texttt{alpha}\) (where p=Inf, 1 or 2) when normType=NORM\_INF, NORM\_L1, or NORM\_L2, respectively; or so that \(\min \_I \texttt{dst} (I)= \texttt{alpha} , \, \, \max \_I \texttt{dst} (I)= \texttt{beta}\) when normType=NORM\_MINMAX (for dense arrays only). The optional mask specifies a sub-array to be normalized. This means that the norm or min-n-max are calculated over the sub-array, and then this sub-array is modified to be normalized. If you want to only use the mask to calculate the norm or min-max but modify the whole array, you can use norm and Mat::convertTo. In case of sparse matrices, only the non-zero values are analyzed and transformed. Because of this, the range transformation for sparse matrices is not allowed since it can shift the zero level. Possible usage with some positive example data: vector<double> positiveData = { 2.0, 8.0, 10.0 }; vector<double> normalizedData\_l1, normalizedData\_l2, normalizedData\_inf, normalizedData\_minmax; // Norm to probability (total count) // sum(numbers) = 20.0 // 2.0 0.1 (2.0/20.0) // 8.0 0.4 (8.0/20.0) // 10.0 0.5 (10.0/20.0) normalize(positiveData, normalizedData\_l1, 1.0, 0.0, NORM\_L1); // Norm to unit vector: ||positiveData|| = 1.0 // 2.0 0.15 // 8.0 0.62 // 10.0 0.77 normalize(positiveData, normalizedData\_l2, 1.0, 0.0, NORM\_L2); // Norm to max element // 2.0 0.2 (2.0/10.0) // 8.0 0.8 (8.0/10.0) // 10.0 1.0 (10.0/10.0) normalize(positiveData, normalizedData\_inf, 1.0, 0.0, NORM\_INF); // Norm to range [0.0;1.0] // 2.0 0.0 (shift to left border) // 8.0 0.75 (6.0/8.0) // 10.0 1.0 (shift to right border) normalize(positiveData, normalizedData\_minmax, 1.0, 0.0, NORM\_MINMAX);Parameters:src - input array.dst - output array of the same size as src .alpha - norm value to normalize to or the lower range boundary in case of the range normalization. normalization. number of channels as src and the depth =CV\_MAT\_DEPTH(dtype). SEE: norm, Mat::convertTo, SparseMat::convertTo

#### normalize public static void normalize([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double alpha, double beta) Normalizes the norm or value range of an array. The function cv::normalize normalizes scale and shift the input array elements so that \(\| \texttt{dst} \| \_{L\_p}= \texttt{alpha}\) (where p=Inf, 1 or 2) when normType=NORM\_INF, NORM\_L1, or NORM\_L2, respectively; or so that \(\min \_I \texttt{dst} (I)= \texttt{alpha} , \, \, \max \_I \texttt{dst} (I)= \texttt{beta}\) when normType=NORM\_MINMAX (for dense arrays only). The optional mask specifies a sub-array to be normalized. This means that the norm or min-n-max are calculated over the sub-array, and then this sub-array is modified to be normalized. If you want to only use the mask to calculate the norm or min-max but modify the whole array, you can use norm and Mat::convertTo. In case of sparse matrices, only the non-zero values are analyzed and transformed. Because of this, the range transformation for sparse matrices is not allowed since it can shift the zero level. Possible usage with some positive example data: vector<double> positiveData = { 2.0, 8.0, 10.0 }; vector<double> normalizedData\_l1, normalizedData\_l2, normalizedData\_inf, normalizedData\_minmax; // Norm to probability (total count) // sum(numbers) = 20.0 // 2.0 0.1 (2.0/20.0) // 8.0 0.4 (8.0/20.0) // 10.0 0.5 (10.0/20.0) normalize(positiveData, normalizedData\_l1, 1.0, 0.0, NORM\_L1); // Norm to unit vector: ||positiveData|| = 1.0 // 2.0 0.15 // 8.0 0.62 // 10.0 0.77 normalize(positiveData, normalizedData\_l2, 1.0, 0.0, NORM\_L2); // Norm to max element // 2.0 0.2 (2.0/10.0) // 8.0 0.8 (8.0/10.0) // 10.0 1.0 (10.0/10.0) normalize(positiveData, normalizedData\_inf, 1.0, 0.0, NORM\_INF); // Norm to range [0.0;1.0] // 2.0 0.0 (shift to left border) // 8.0 0.75 (6.0/8.0) // 10.0 1.0 (shift to right border) normalize(positiveData, normalizedData\_minmax, 1.0, 0.0, NORM\_MINMAX);Parameters:src - input array.dst - output array of the same size as src .alpha - norm value to normalize to or the lower range boundary in case of the range normalization.beta - upper range boundary in case of the range normalization; it is not used for the norm normalization. number of channels as src and the depth =CV\_MAT\_DEPTH(dtype). SEE: norm, Mat::convertTo, SparseMat::convertTo

#### normalize public static void normalize([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double alpha, double beta, int norm\_type) Normalizes the norm or value range of an array. The function cv::normalize normalizes scale and shift the input array elements so that \(\| \texttt{dst} \| \_{L\_p}= \texttt{alpha}\) (where p=Inf, 1 or 2) when normType=NORM\_INF, NORM\_L1, or NORM\_L2, respectively; or so that \(\min \_I \texttt{dst} (I)= \texttt{alpha} , \, \, \max \_I \texttt{dst} (I)= \texttt{beta}\) when normType=NORM\_MINMAX (for dense arrays only). The optional mask specifies a sub-array to be normalized. This means that the norm or min-n-max are calculated over the sub-array, and then this sub-array is modified to be normalized. If you want to only use the mask to calculate the norm or min-max but modify the whole array, you can use norm and Mat::convertTo. In case of sparse matrices, only the non-zero values are analyzed and transformed. Because of this, the range transformation for sparse matrices is not allowed since it can shift the zero level. Possible usage with some positive example data: vector<double> positiveData = { 2.0, 8.0, 10.0 }; vector<double> normalizedData\_l1, normalizedData\_l2, normalizedData\_inf, normalizedData\_minmax; // Norm to probability (total count) // sum(numbers) = 20.0 // 2.0 0.1 (2.0/20.0) // 8.0 0.4 (8.0/20.0) // 10.0 0.5 (10.0/20.0) normalize(positiveData, normalizedData\_l1, 1.0, 0.0, NORM\_L1); // Norm to unit vector: ||positiveData|| = 1.0 // 2.0 0.15 // 8.0 0.62 // 10.0 0.77 normalize(positiveData, normalizedData\_l2, 1.0, 0.0, NORM\_L2); // Norm to max element // 2.0 0.2 (2.0/10.0) // 8.0 0.8 (8.0/10.0) // 10.0 1.0 (10.0/10.0) normalize(positiveData, normalizedData\_inf, 1.0, 0.0, NORM\_INF); // Norm to range [0.0;1.0] // 2.0 0.0 (shift to left border) // 8.0 0.75 (6.0/8.0) // 10.0 1.0 (shift to right border) normalize(positiveData, normalizedData\_minmax, 1.0, 0.0, NORM\_MINMAX);Parameters:src - input array.dst - output array of the same size as src .alpha - norm value to normalize to or the lower range boundary in case of the range normalization.beta - upper range boundary in case of the range normalization; it is not used for the norm normalization.norm\_type - normalization type (see cv::NormTypes). number of channels as src and the depth =CV\_MAT\_DEPTH(dtype). SEE: norm, Mat::convertTo, SparseMat::convertTo

#### normalize public static void normalize([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double alpha, double beta, int norm\_type, int dtype) Normalizes the norm or value range of an array. The function cv::normalize normalizes scale and shift the input array elements so that \(\| \texttt{dst} \| \_{L\_p}= \texttt{alpha}\) (where p=Inf, 1 or 2) when normType=NORM\_INF, NORM\_L1, or NORM\_L2, respectively; or so that \(\min \_I \texttt{dst} (I)= \texttt{alpha} , \, \, \max \_I \texttt{dst} (I)= \texttt{beta}\) when normType=NORM\_MINMAX (for dense arrays only). The optional mask specifies a sub-array to be normalized. This means that the norm or min-n-max are calculated over the sub-array, and then this sub-array is modified to be normalized. If you want to only use the mask to calculate the norm or min-max but modify the whole array, you can use norm and Mat::convertTo. In case of sparse matrices, only the non-zero values are analyzed and transformed. Because of this, the range transformation for sparse matrices is not allowed since it can shift the zero level. Possible usage with some positive example data: vector<double> positiveData = { 2.0, 8.0, 10.0 }; vector<double> normalizedData\_l1, normalizedData\_l2, normalizedData\_inf, normalizedData\_minmax; // Norm to probability (total count) // sum(numbers) = 20.0 // 2.0 0.1 (2.0/20.0) // 8.0 0.4 (8.0/20.0) // 10.0 0.5 (10.0/20.0) normalize(positiveData, normalizedData\_l1, 1.0, 0.0, NORM\_L1); // Norm to unit vector: ||positiveData|| = 1.0 // 2.0 0.15 // 8.0 0.62 // 10.0 0.77 normalize(positiveData, normalizedData\_l2, 1.0, 0.0, NORM\_L2); // Norm to max element // 2.0 0.2 (2.0/10.0) // 8.0 0.8 (8.0/10.0) // 10.0 1.0 (10.0/10.0) normalize(positiveData, normalizedData\_inf, 1.0, 0.0, NORM\_INF); // Norm to range [0.0;1.0] // 2.0 0.0 (shift to left border) // 8.0 0.75 (6.0/8.0) // 10.0 1.0 (shift to right border) normalize(positiveData, normalizedData\_minmax, 1.0, 0.0, NORM\_MINMAX);Parameters:src - input array.dst - output array of the same size as src .alpha - norm value to normalize to or the lower range boundary in case of the range normalization.beta - upper range boundary in case of the range normalization; it is not used for the norm normalization.norm\_type - normalization type (see cv::NormTypes).dtype - when negative, the output array has the same type as src; otherwise, it has the same number of channels as src and the depth =CV\_MAT\_DEPTH(dtype). SEE: norm, Mat::convertTo, SparseMat::convertTo

#### normalize public static void normalize([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double alpha, double beta, int norm\_type, int dtype, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) Normalizes the norm or value range of an array. The function cv::normalize normalizes scale and shift the input array elements so that \(\| \texttt{dst} \| \_{L\_p}= \texttt{alpha}\) (where p=Inf, 1 or 2) when normType=NORM\_INF, NORM\_L1, or NORM\_L2, respectively; or so that \(\min \_I \texttt{dst} (I)= \texttt{alpha} , \, \, \max \_I \texttt{dst} (I)= \texttt{beta}\) when normType=NORM\_MINMAX (for dense arrays only). The optional mask specifies a sub-array to be normalized. This means that the norm or min-n-max are calculated over the sub-array, and then this sub-array is modified to be normalized. If you want to only use the mask to calculate the norm or min-max but modify the whole array, you can use norm and Mat::convertTo. In case of sparse matrices, only the non-zero values are analyzed and transformed. Because of this, the range transformation for sparse matrices is not allowed since it can shift the zero level. Possible usage with some positive example data: vector<double> positiveData = { 2.0, 8.0, 10.0 }; vector<double> normalizedData\_l1, normalizedData\_l2, normalizedData\_inf, normalizedData\_minmax; // Norm to probability (total count) // sum(numbers) = 20.0 // 2.0 0.1 (2.0/20.0) // 8.0 0.4 (8.0/20.0) // 10.0 0.5 (10.0/20.0) normalize(positiveData, normalizedData\_l1, 1.0, 0.0, NORM\_L1); // Norm to unit vector: ||positiveData|| = 1.0 // 2.0 0.15 // 8.0 0.62 // 10.0 0.77 normalize(positiveData, normalizedData\_l2, 1.0, 0.0, NORM\_L2); // Norm to max element // 2.0 0.2 (2.0/10.0) // 8.0 0.8 (8.0/10.0) // 10.0 1.0 (10.0/10.0) normalize(positiveData, normalizedData\_inf, 1.0, 0.0, NORM\_INF); // Norm to range [0.0;1.0] // 2.0 0.0 (shift to left border) // 8.0 0.75 (6.0/8.0) // 10.0 1.0 (shift to right border) normalize(positiveData, normalizedData\_minmax, 1.0, 0.0, NORM\_MINMAX);Parameters:src - input array.dst - output array of the same size as src .alpha - norm value to normalize to or the lower range boundary in case of the range normalization.beta - upper range boundary in case of the range normalization; it is not used for the norm normalization.norm\_type - normalization type (see cv::NormTypes).dtype - when negative, the output array has the same type as src; otherwise, it has the same number of channels as src and the depth =CV\_MAT\_DEPTH(dtype).mask - optional operation mask. SEE: norm, Mat::convertTo, SparseMat::convertTo

#### patchNaNs public static void patchNaNs([Mat](http://docs.google.com/org/opencv/core/Mat.html) a) converts NaNs to the given numberParameters:a - input/output matrix (CV\_32F type).

#### patchNaNs public static void patchNaNs([Mat](http://docs.google.com/org/opencv/core/Mat.html) a, double val) converts NaNs to the given numberParameters:a - input/output matrix (CV\_32F type).val - value to convert the NaNs

#### PCABackProject public static void PCABackProject([Mat](http://docs.google.com/org/opencv/core/Mat.html) data, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mean, [Mat](http://docs.google.com/org/opencv/core/Mat.html) eigenvectors, [Mat](http://docs.google.com/org/opencv/core/Mat.html) result) wrap PCA::backProjectParameters:data - automatically generatedmean - automatically generatedeigenvectors - automatically generatedresult - automatically generated

#### PCACompute public static void PCACompute([Mat](http://docs.google.com/org/opencv/core/Mat.html) data, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mean, [Mat](http://docs.google.com/org/opencv/core/Mat.html) eigenvectors) wrap PCA::operator()Parameters:data - automatically generatedmean - automatically generatedeigenvectors - automatically generated

#### PCACompute public static void PCACompute([Mat](http://docs.google.com/org/opencv/core/Mat.html) data, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mean, [Mat](http://docs.google.com/org/opencv/core/Mat.html) eigenvectors, double retainedVariance) wrap PCA::operator()Parameters:data - automatically generatedmean - automatically generatedeigenvectors - automatically generatedretainedVariance - automatically generated

#### PCACompute public static void PCACompute([Mat](http://docs.google.com/org/opencv/core/Mat.html) data, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mean, [Mat](http://docs.google.com/org/opencv/core/Mat.html) eigenvectors, int maxComponents) wrap PCA::operator()Parameters:data - automatically generatedmean - automatically generatedeigenvectors - automatically generatedmaxComponents - automatically generated

#### PCACompute2 public static void PCACompute2([Mat](http://docs.google.com/org/opencv/core/Mat.html) data, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mean, [Mat](http://docs.google.com/org/opencv/core/Mat.html) eigenvectors, [Mat](http://docs.google.com/org/opencv/core/Mat.html) eigenvalues) wrap PCA::operator() and add eigenvalues output parameterParameters:data - automatically generatedmean - automatically generatedeigenvectors - automatically generatedeigenvalues - automatically generated

#### PCACompute2 public static void PCACompute2([Mat](http://docs.google.com/org/opencv/core/Mat.html) data, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mean, [Mat](http://docs.google.com/org/opencv/core/Mat.html) eigenvectors, [Mat](http://docs.google.com/org/opencv/core/Mat.html) eigenvalues, double retainedVariance) wrap PCA::operator() and add eigenvalues output parameterParameters:data - automatically generatedmean - automatically generatedeigenvectors - automatically generatedeigenvalues - automatically generatedretainedVariance - automatically generated

#### PCACompute2 public static void PCACompute2([Mat](http://docs.google.com/org/opencv/core/Mat.html) data, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mean, [Mat](http://docs.google.com/org/opencv/core/Mat.html) eigenvectors, [Mat](http://docs.google.com/org/opencv/core/Mat.html) eigenvalues, int maxComponents) wrap PCA::operator() and add eigenvalues output parameterParameters:data - automatically generatedmean - automatically generatedeigenvectors - automatically generatedeigenvalues - automatically generatedmaxComponents - automatically generated

#### PCAProject public static void PCAProject([Mat](http://docs.google.com/org/opencv/core/Mat.html) data, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mean, [Mat](http://docs.google.com/org/opencv/core/Mat.html) eigenvectors, [Mat](http://docs.google.com/org/opencv/core/Mat.html) result) wrap PCA::projectParameters:data - automatically generatedmean - automatically generatedeigenvectors - automatically generatedresult - automatically generated

#### perspectiveTransform public static void perspectiveTransform([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) m) Performs the perspective matrix transformation of vectors. The function cv::perspectiveTransform transforms every element of src by treating it as a 2D or 3D vector, in the following way: \((x, y, z) \rightarrow (x'/w, y'/w, z'/w)\) where \((x', y', z', w') = \texttt{mat} \cdot \begin{bmatrix} x & y & z & 1 \end{bmatrix}\) and \(w = \fork{w'}{if \(w' \ne 0\)}{\infty}{otherwise}\) Here a 3D vector transformation is shown. In case of a 2D vector transformation, the z component is omitted. **Note:** The function transforms a sparse set of 2D or 3D vectors. If you want to transform an image using perspective transformation, use warpPerspective . If you have an inverse problem, that is, you want to compute the most probable perspective transformation out of several pairs of corresponding points, you can use getPerspectiveTransform or findHomography .Parameters:src - input two-channel or three-channel floating-point array; each element is a 2D/3D vector to be transformed.dst - output array of the same size and type as src.m - 3x3 or 4x4 floating-point transformation matrix. SEE: transform, warpPerspective, getPerspectiveTransform, findHomography

#### phase public static void phase([Mat](http://docs.google.com/org/opencv/core/Mat.html) x, [Mat](http://docs.google.com/org/opencv/core/Mat.html) y, [Mat](http://docs.google.com/org/opencv/core/Mat.html) angle) Calculates the rotation angle of 2D vectors. The function cv::phase calculates the rotation angle of each 2D vector that is formed from the corresponding elements of x and y : \(\texttt{angle} (I) = \texttt{atan2} ( \texttt{y} (I), \texttt{x} (I))\) The angle estimation accuracy is about 0.3 degrees. When x(I)=y(I)=0 , the corresponding angle(I) is set to 0.Parameters:x - input floating-point array of x-coordinates of 2D vectors.y - input array of y-coordinates of 2D vectors; it must have the same size and the same type as x.angle - output array of vector angles; it has the same size and same type as x . degrees, otherwise, they are measured in radians.

#### phase public static void phase([Mat](http://docs.google.com/org/opencv/core/Mat.html) x, [Mat](http://docs.google.com/org/opencv/core/Mat.html) y, [Mat](http://docs.google.com/org/opencv/core/Mat.html) angle, boolean angleInDegrees) Calculates the rotation angle of 2D vectors. The function cv::phase calculates the rotation angle of each 2D vector that is formed from the corresponding elements of x and y : \(\texttt{angle} (I) = \texttt{atan2} ( \texttt{y} (I), \texttt{x} (I))\) The angle estimation accuracy is about 0.3 degrees. When x(I)=y(I)=0 , the corresponding angle(I) is set to 0.Parameters:x - input floating-point array of x-coordinates of 2D vectors.y - input array of y-coordinates of 2D vectors; it must have the same size and the same type as x.angle - output array of vector angles; it has the same size and same type as x .angleInDegrees - when true, the function calculates the angle in degrees, otherwise, they are measured in radians.

#### polarToCart public static void polarToCart([Mat](http://docs.google.com/org/opencv/core/Mat.html) magnitude, [Mat](http://docs.google.com/org/opencv/core/Mat.html) angle, [Mat](http://docs.google.com/org/opencv/core/Mat.html) x, [Mat](http://docs.google.com/org/opencv/core/Mat.html) y) Calculates x and y coordinates of 2D vectors from their magnitude and angle. The function cv::polarToCart calculates the Cartesian coordinates of each 2D vector represented by the corresponding elements of magnitude and angle: \(\begin{array}{l} \texttt{x} (I) = \texttt{magnitude} (I) \cos ( \texttt{angle} (I)) \\ \texttt{y} (I) = \texttt{magnitude} (I) \sin ( \texttt{angle} (I)) \\ \end{array}\) The relative accuracy of the estimated coordinates is about 1e-6.Parameters:magnitude - input floating-point array of magnitudes of 2D vectors; it can be an empty matrix (=Mat()), in this case, the function assumes that all the magnitudes are =1; if it is not empty, it must have the same size and type as angle.angle - input floating-point array of angles of 2D vectors.x - output array of x-coordinates of 2D vectors; it has the same size and type as angle.y - output array of y-coordinates of 2D vectors; it has the same size and type as angle. degrees, otherwise, they are measured in radians. SEE: cartToPolar, magnitude, phase, exp, log, pow, sqrt

#### polarToCart public static void polarToCart([Mat](http://docs.google.com/org/opencv/core/Mat.html) magnitude, [Mat](http://docs.google.com/org/opencv/core/Mat.html) angle, [Mat](http://docs.google.com/org/opencv/core/Mat.html) x, [Mat](http://docs.google.com/org/opencv/core/Mat.html) y, boolean angleInDegrees) Calculates x and y coordinates of 2D vectors from their magnitude and angle. The function cv::polarToCart calculates the Cartesian coordinates of each 2D vector represented by the corresponding elements of magnitude and angle: \(\begin{array}{l} \texttt{x} (I) = \texttt{magnitude} (I) \cos ( \texttt{angle} (I)) \\ \texttt{y} (I) = \texttt{magnitude} (I) \sin ( \texttt{angle} (I)) \\ \end{array}\) The relative accuracy of the estimated coordinates is about 1e-6.Parameters:magnitude - input floating-point array of magnitudes of 2D vectors; it can be an empty matrix (=Mat()), in this case, the function assumes that all the magnitudes are =1; if it is not empty, it must have the same size and type as angle.angle - input floating-point array of angles of 2D vectors.x - output array of x-coordinates of 2D vectors; it has the same size and type as angle.y - output array of y-coordinates of 2D vectors; it has the same size and type as angle.angleInDegrees - when true, the input angles are measured in degrees, otherwise, they are measured in radians. SEE: cartToPolar, magnitude, phase, exp, log, pow, sqrt

#### pow public static void pow([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, double power, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Raises every array element to a power. The function cv::pow raises every element of the input array to power : \(\texttt{dst} (I) = \fork{\texttt{src}(I)^{power}}{if \(\texttt{power}\) is integer}{|\texttt{src}(I)|^{power}}{otherwise}\) So, for a non-integer power exponent, the absolute values of input array elements are used. However, it is possible to get true values for negative values using some extra operations. In the example below, computing the 5th root of array src shows: Mat mask = src < 0; pow(src, 1./5, dst); subtract(Scalar::all(0), dst, dst, mask); For some values of power, such as integer values, 0.5 and -0.5, specialized faster algorithms are used. Special values (NaN, Inf) are not handled.Parameters:src - input array.power - exponent of power.dst - output array of the same size and type as src. SEE: sqrt, exp, log, cartToPolar, polarToCart

#### PSNR public static double PSNR([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2) Computes the Peak Signal-to-Noise Ratio (PSNR) image quality metric. This function calculates the Peak Signal-to-Noise Ratio (PSNR) image quality metric in decibels (dB), between two input arrays src1 and src2. Arrays must have depth CV\_8U. The PSNR is calculated as follows: \( \texttt{PSNR} = 10 \cdot \log\_{10}{\left( \frac{R^2}{MSE} \right) } \) where R is the maximum integer value of depth CV\_8U (255) and MSE is the mean squared error between the two arrays.Parameters:src1 - first input array.src2 - second input array of the same size as src1. Returns:automatically generated

#### randn public static void randn([Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double mean, double stddev) Fills the array with normally distributed random numbers. The function cv::randn fills the matrix dst with normally distributed random numbers with the specified mean vector and the standard deviation matrix. The generated random numbers are clipped to fit the value range of the output array data type.Parameters:dst - output array of random numbers; the array must be pre-allocated and have 1 to 4 channels.mean - mean value (expectation) of the generated random numbers.stddev - standard deviation of the generated random numbers; it can be either a vector (in which case a diagonal standard deviation matrix is assumed) or a square matrix. SEE: RNG, randu

#### randShuffle public static void randShuffle([Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Shuffles the array elements randomly. The function cv::randShuffle shuffles the specified 1D array by randomly choosing pairs of elements and swapping them. The number of such swap operations will be dst.rows\\*dst.cols\\*iterFactor .Parameters:dst - input/output numerical 1D array. below). instead. SEE: RNG, sort

#### randShuffle public static void randShuffle([Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double iterFactor) Shuffles the array elements randomly. The function cv::randShuffle shuffles the specified 1D array by randomly choosing pairs of elements and swapping them. The number of such swap operations will be dst.rows\\*dst.cols\\*iterFactor .Parameters:dst - input/output numerical 1D array.iterFactor - scale factor that determines the number of random swap operations (see the details below). instead. SEE: RNG, sort

#### randu public static void randu([Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, double low, double high) Generates a single uniformly-distributed random number or an array of random numbers. Non-template variant of the function fills the matrix dst with uniformly-distributed random numbers from the specified range: \(\texttt{low} \_c \leq \texttt{dst} (I)\_c < \texttt{high} \_c\)Parameters:dst - output array of random numbers; the array must be pre-allocated.low - inclusive lower boundary of the generated random numbers.high - exclusive upper boundary of the generated random numbers. SEE: RNG, randn, theRNG

#### reduce public static void reduce([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int dim, int rtype) Reduces a matrix to a vector. The function #reduce reduces the matrix to a vector by treating the matrix rows/columns as a set of 1D vectors and performing the specified operation on the vectors until a single row/column is obtained. For example, the function can be used to compute horizontal and vertical projections of a raster image. In case of #REDUCE\_MAX and #REDUCE\_MIN , the output image should have the same type as the source one. In case of #REDUCE\_SUM and #REDUCE\_AVG , the output may have a larger element bit-depth to preserve accuracy. And multi-channel arrays are also supported in these two reduction modes. The following code demonstrates its usage for a single channel matrix. SNIPPET: snippets/core\_reduce.cpp example And the following code demonstrates its usage for a two-channel matrix. SNIPPET: snippets/core\_reduce.cpp example2Parameters:src - input 2D matrix.dst - output vector. Its size and type is defined by dim and dtype parameters.dim - dimension index along which the matrix is reduced. 0 means that the matrix is reduced to a single row. 1 means that the matrix is reduced to a single column.rtype - reduction operation that could be one of #ReduceTypes otherwise, its type will be CV\_MAKE\_TYPE(CV\_MAT\_DEPTH(dtype), src.channels()). SEE: repeat

#### reduce public static void reduce([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int dim, int rtype, int dtype) Reduces a matrix to a vector. The function #reduce reduces the matrix to a vector by treating the matrix rows/columns as a set of 1D vectors and performing the specified operation on the vectors until a single row/column is obtained. For example, the function can be used to compute horizontal and vertical projections of a raster image. In case of #REDUCE\_MAX and #REDUCE\_MIN , the output image should have the same type as the source one. In case of #REDUCE\_SUM and #REDUCE\_AVG , the output may have a larger element bit-depth to preserve accuracy. And multi-channel arrays are also supported in these two reduction modes. The following code demonstrates its usage for a single channel matrix. SNIPPET: snippets/core\_reduce.cpp example And the following code demonstrates its usage for a two-channel matrix. SNIPPET: snippets/core\_reduce.cpp example2Parameters:src - input 2D matrix.dst - output vector. Its size and type is defined by dim and dtype parameters.dim - dimension index along which the matrix is reduced. 0 means that the matrix is reduced to a single row. 1 means that the matrix is reduced to a single column.rtype - reduction operation that could be one of #ReduceTypesdtype - when negative, the output vector will have the same type as the input matrix, otherwise, its type will be CV\_MAKE\_TYPE(CV\_MAT\_DEPTH(dtype), src.channels()). SEE: repeat

#### repeat public static void repeat([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, int ny, int nx, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Fills the output array with repeated copies of the input array. The function cv::repeat duplicates the input array one or more times along each of the two axes: \(\texttt{dst} \_{ij}= \texttt{src} \_{i\mod src.rows, \; j\mod src.cols }\) The second variant of the function is more convenient to use with REF: MatrixExpressions.Parameters:src - input array to replicate.ny - Flag to specify how many times the src is repeated along the vertical axis.nx - Flag to specify how many times the src is repeated along the horizontal axis.dst - output array of the same type as src. SEE: cv::reduce

#### rotate public static void rotate([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int rotateCode) Rotates a 2D array in multiples of 90 degrees. The function cv::rotate rotates the array in one of three different ways: Rotate by 90 degrees clockwise (rotateCode = ROTATE\_90\_CLOCKWISE). Rotate by 180 degrees clockwise (rotateCode = ROTATE\_180). Rotate by 270 degrees clockwise (rotateCode = ROTATE\_90\_COUNTERCLOCKWISE).Parameters:src - input array.dst - output array of the same type as src. The size is the same with ROTATE\_180, and the rows and cols are switched for ROTATE\_90\_CLOCKWISE and ROTATE\_90\_COUNTERCLOCKWISE.rotateCode - an enum to specify how to rotate the array; see the enum #RotateFlags SEE: transpose , repeat , completeSymm, flip, RotateFlags

#### scaleAdd public static void scaleAdd([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, double alpha, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Calculates the sum of a scaled array and another array. The function scaleAdd is one of the classical primitive linear algebra operations, known as DAXPY or SAXPY in [BLAS](http://en.wikipedia.org/wiki/Basic\_Linear\_Algebra\_Subprograms). It calculates the sum of a scaled array and another array: \(\texttt{dst} (I)= \texttt{scale} \cdot \texttt{src1} (I) + \texttt{src2} (I)\) The function can also be emulated with a matrix expression, for example: Mat A(3, 3, CV\_64F); ... A.row(0) = A.row(1)\*2 + A.row(2);Parameters:src1 - first input array.alpha - scale factor for the first array.src2 - second input array of the same size and type as src1.dst - output array of the same size and type as src1. SEE: add, addWeighted, subtract, Mat::dot, Mat::convertTo

#### setErrorVerbosity public static void setErrorVerbosity(boolean verbose)

#### setIdentity public static void setIdentity([Mat](http://docs.google.com/org/opencv/core/Mat.html) mtx) Initializes a scaled identity matrix. The function cv::setIdentity initializes a scaled identity matrix: \(\texttt{mtx} (i,j)= \fork{\texttt{value}}{ if \(i=j\)}{0}{otherwise}\) The function can also be emulated using the matrix initializers and the matrix expressions: Mat A = Mat::eye(4, 3, CV\_32F)\*5; // A will be set to [[5, 0, 0], [0, 5, 0], [0, 0, 5], [0, 0, 0]]Parameters:mtx - matrix to initialize (not necessarily square). SEE: Mat::zeros, Mat::ones, Mat::setTo, Mat::operator=

#### setIdentity public static void setIdentity([Mat](http://docs.google.com/org/opencv/core/Mat.html) mtx, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) s) Initializes a scaled identity matrix. The function cv::setIdentity initializes a scaled identity matrix: \(\texttt{mtx} (i,j)= \fork{\texttt{value}}{ if \(i=j\)}{0}{otherwise}\) The function can also be emulated using the matrix initializers and the matrix expressions: Mat A = Mat::eye(4, 3, CV\_32F)\*5; // A will be set to [[5, 0, 0], [0, 5, 0], [0, 0, 5], [0, 0, 0]]Parameters:mtx - matrix to initialize (not necessarily square).s - value to assign to diagonal elements. SEE: Mat::zeros, Mat::ones, Mat::setTo, Mat::operator=

#### setNumThreads public static void setNumThreads(int nthreads)

OpenCV will try to set the number of threads for the next parallel region. If threads == 0, OpenCV will disable threading optimizations and run all it's functions sequentially. Passing threads < 0 will reset threads number to system default. This function must be called outside of parallel region. OpenCV will try to run its functions with specified threads number, but some behaviour differs from framework:

* + TBB - User-defined parallel constructions will run with the same threads number, if another is not specified. If later on user creates his own scheduler, OpenCV will use it.
  + OpenMP - No special defined behaviour.
  + Concurrency - If threads == 1, OpenCV will disable threading optimizations and run its functions sequentially.
  + GCD - Supports only values <= 0.
  + C= - No special defined behaviour.

Parameters:nthreads - Number of threads used by OpenCV. SEE: getNumThreads, getThreadNum

#### setRNGSeed public static void setRNGSeed(int seed) Sets state of default random number generator. The function cv::setRNGSeed sets state of default random number generator to custom value.Parameters:seed - new state for default random number generator SEE: RNG, randu, randn

#### setUseIPP\_NE public static void setUseIPP\_NE(boolean flag)

#### setUseIPP\_NotExact public static void setUseIPP\_NotExact(boolean flag)

#### setUseIPP public static void setUseIPP(boolean flag)

#### solve public static boolean solve([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Solves one or more linear systems or least-squares problems. The function cv::solve solves a linear system or least-squares problem (the latter is possible with SVD or QR methods, or by specifying the flag #DECOMP\_NORMAL ): \(\texttt{dst} = \arg \min \_X \| \texttt{src1} \cdot \texttt{X} - \texttt{src2} \|\) If #DECOMP\_LU or #DECOMP\_CHOLESKY method is used, the function returns 1 if src1 (or \(\texttt{src1}^T\texttt{src1}\) ) is non-singular. Otherwise, it returns 0. In the latter case, dst is not valid. Other methods find a pseudo-solution in case of a singular left-hand side part. **Note:** If you want to find a unity-norm solution of an under-defined singular system \(\texttt{src1}\cdot\texttt{dst}=0\) , the function solve will not do the work. Use SVD::solveZ instead.Parameters:src1 - input matrix on the left-hand side of the system.src2 - input matrix on the right-hand side of the system.dst - output solution. SEE: invert, SVD, eigen Returns:automatically generated

#### solve public static boolean solve([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int flags) Solves one or more linear systems or least-squares problems. The function cv::solve solves a linear system or least-squares problem (the latter is possible with SVD or QR methods, or by specifying the flag #DECOMP\_NORMAL ): \(\texttt{dst} = \arg \min \_X \| \texttt{src1} \cdot \texttt{X} - \texttt{src2} \|\) If #DECOMP\_LU or #DECOMP\_CHOLESKY method is used, the function returns 1 if src1 (or \(\texttt{src1}^T\texttt{src1}\) ) is non-singular. Otherwise, it returns 0. In the latter case, dst is not valid. Other methods find a pseudo-solution in case of a singular left-hand side part. **Note:** If you want to find a unity-norm solution of an under-defined singular system \(\texttt{src1}\cdot\texttt{dst}=0\) , the function solve will not do the work. Use SVD::solveZ instead.Parameters:src1 - input matrix on the left-hand side of the system.src2 - input matrix on the right-hand side of the system.dst - output solution.flags - solution (matrix inversion) method (#DecompTypes) SEE: invert, SVD, eigen Returns:automatically generated

#### solveCubic public static int solveCubic([Mat](http://docs.google.com/org/opencv/core/Mat.html) coeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) roots)

Finds the real roots of a cubic equation. The function solveCubic finds the real roots of a cubic equation:

* + if coeffs is a 4-element vector: \(\texttt{coeffs} [0] x^3 + \texttt{coeffs} [1] x^2 + \texttt{coeffs} [2] x + \texttt{coeffs} [3] = 0\)
  + if coeffs is a 3-element vector: \(x^3 + \texttt{coeffs} [0] x^2 + \texttt{coeffs} [1] x + \texttt{coeffs} [2] = 0\)

The roots are stored in the roots array.Parameters:coeffs - equation coefficients, an array of 3 or 4 elements.roots - output array of real roots that has 1 or 3 elements. Returns:number of real roots. It can be 0, 1 or 2.

#### solvePoly public static double solvePoly([Mat](http://docs.google.com/org/opencv/core/Mat.html) coeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) roots) Finds the real or complex roots of a polynomial equation. The function cv::solvePoly finds real and complex roots of a polynomial equation: \(\texttt{coeffs} [n] x^{n} + \texttt{coeffs} [n-1] x^{n-1} + ... + \texttt{coeffs} [1] x + \texttt{coeffs} [0] = 0\)Parameters:coeffs - array of polynomial coefficients.roots - output (complex) array of roots. Returns:automatically generated

#### solvePoly public static double solvePoly([Mat](http://docs.google.com/org/opencv/core/Mat.html) coeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) roots, int maxIters) Finds the real or complex roots of a polynomial equation. The function cv::solvePoly finds real and complex roots of a polynomial equation: \(\texttt{coeffs} [n] x^{n} + \texttt{coeffs} [n-1] x^{n-1} + ... + \texttt{coeffs} [1] x + \texttt{coeffs} [0] = 0\)Parameters:coeffs - array of polynomial coefficients.roots - output (complex) array of roots.maxIters - maximum number of iterations the algorithm does. Returns:automatically generated

#### sort public static void sort([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int flags) Sorts each row or each column of a matrix. The function cv::sort sorts each matrix row or each matrix column in ascending or descending order. So you should pass two operation flags to get desired behaviour. If you want to sort matrix rows or columns lexicographically, you can use STL std::sort generic function with the proper comparison predicate.Parameters:src - input single-channel array.dst - output array of the same size and type as src.flags - operation flags, a combination of #SortFlags SEE: sortIdx, randShuffle

#### sortIdx public static void sortIdx([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, int flags) Sorts each row or each column of a matrix. The function cv::sortIdx sorts each matrix row or each matrix column in the ascending or descending order. So you should pass two operation flags to get desired behaviour. Instead of reordering the elements themselves, it stores the indices of sorted elements in the output array. For example: Mat A = Mat::eye(3,3,CV\_32F), B; sortIdx(A, B, SORT\_EVERY\_ROW + SORT\_ASCENDING); // B will probably contain // (because of equal elements in A some permutations are possible): // [[1, 2, 0], [0, 2, 1], [0, 1, 2]]Parameters:src - input single-channel array.dst - output integer array of the same size as src.flags - operation flags that could be a combination of cv::SortFlags SEE: sort, randShuffle

#### split public static void split([Mat](http://docs.google.com/org/opencv/core/Mat.html) m, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> mv)Parameters:m - input multi-channel array.mv - output vector of arrays; the arrays themselves are reallocated, if needed.

#### sqrt public static void sqrt([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Calculates a square root of array elements. The function cv::sqrt calculates a square root of each input array element. In case of multi-channel arrays, each channel is processed independently. The accuracy is approximately the same as of the built-in std::sqrt .Parameters:src - input floating-point array.dst - output array of the same size and type as src.

#### subtract public static void subtract([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst)

Calculates the per-element difference between two arrays or array and a scalar. The function subtract calculates:

* + Difference between two arrays, when both input arrays have the same size and the same number of channels: \(\texttt{dst}(I) = \texttt{saturate} ( \texttt{src1}(I) - \texttt{src2}(I)) \quad \texttt{if mask}(I) \ne0\)
  + Difference between an array and a scalar, when src2 is constructed from Scalar or has the same number of elements as src1.channels(): \(\texttt{dst}(I) = \texttt{saturate} ( \texttt{src1}(I) - \texttt{src2} ) \quad \texttt{if mask}(I) \ne0\)
  + Difference between a scalar and an array, when src1 is constructed from Scalar or has the same number of elements as src2.channels(): \(\texttt{dst}(I) = \texttt{saturate} ( \texttt{src1} - \texttt{src2}(I) ) \quad \texttt{if mask}(I) \ne0\)
  + The reverse difference between a scalar and an array in the case of SubRS: \(\texttt{dst}(I) = \texttt{saturate} ( \texttt{src2} - \texttt{src1}(I) ) \quad \texttt{if mask}(I) \ne0\) where I is a multi-dimensional index of array elements. In case of multi-channel arrays, each channel is processed independently.

The first function in the list above can be replaced with matrix expressions: dst = src1 - src2; dst -= src1; // equivalent to subtract(dst, src1, dst); The input arrays and the output array can all have the same or different depths. For example, you can subtract to 8-bit unsigned arrays and store the difference in a 16-bit signed array. Depth of the output array is determined by dtype parameter. In the second and third cases above, as well as in the first case, when src1.depth() == src2.depth(), dtype can be set to the default -1. In this case the output array will have the same depth as the input array, be it src1, src2 or both. **Note:** Saturation is not applied when the output array has the depth CV\_32S. You may even get result of an incorrect sign in the case of overflow.Parameters:src1 - first input array or a scalar.src2 - second input array or a scalar.dst - output array of the same size and the same number of channels as the input array. of the output array to be changed. SEE: add, addWeighted, scaleAdd, Mat::convertTo

#### subtract public static void subtract([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask)

Calculates the per-element difference between two arrays or array and a scalar. The function subtract calculates:

* + Difference between two arrays, when both input arrays have the same size and the same number of channels: \(\texttt{dst}(I) = \texttt{saturate} ( \texttt{src1}(I) - \texttt{src2}(I)) \quad \texttt{if mask}(I) \ne0\)
  + Difference between an array and a scalar, when src2 is constructed from Scalar or has the same number of elements as src1.channels(): \(\texttt{dst}(I) = \texttt{saturate} ( \texttt{src1}(I) - \texttt{src2} ) \quad \texttt{if mask}(I) \ne0\)
  + Difference between a scalar and an array, when src1 is constructed from Scalar or has the same number of elements as src2.channels(): \(\texttt{dst}(I) = \texttt{saturate} ( \texttt{src1} - \texttt{src2}(I) ) \quad \texttt{if mask}(I) \ne0\)
  + The reverse difference between a scalar and an array in the case of SubRS: \(\texttt{dst}(I) = \texttt{saturate} ( \texttt{src2} - \texttt{src1}(I) ) \quad \texttt{if mask}(I) \ne0\) where I is a multi-dimensional index of array elements. In case of multi-channel arrays, each channel is processed independently.

The first function in the list above can be replaced with matrix expressions: dst = src1 - src2; dst -= src1; // equivalent to subtract(dst, src1, dst); The input arrays and the output array can all have the same or different depths. For example, you can subtract to 8-bit unsigned arrays and store the difference in a 16-bit signed array. Depth of the output array is determined by dtype parameter. In the second and third cases above, as well as in the first case, when src1.depth() == src2.depth(), dtype can be set to the default -1. In this case the output array will have the same depth as the input array, be it src1, src2 or both. **Note:** Saturation is not applied when the output array has the depth CV\_32S. You may even get result of an incorrect sign in the case of overflow.Parameters:src1 - first input array or a scalar.src2 - second input array or a scalar.dst - output array of the same size and the same number of channels as the input array.mask - optional operation mask; this is an 8-bit single channel array that specifies elements of the output array to be changed. SEE: add, addWeighted, scaleAdd, Mat::convertTo

#### subtract public static void subtract([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask, int dtype)

Calculates the per-element difference between two arrays or array and a scalar. The function subtract calculates:

* + Difference between two arrays, when both input arrays have the same size and the same number of channels: \(\texttt{dst}(I) = \texttt{saturate} ( \texttt{src1}(I) - \texttt{src2}(I)) \quad \texttt{if mask}(I) \ne0\)
  + Difference between an array and a scalar, when src2 is constructed from Scalar or has the same number of elements as src1.channels(): \(\texttt{dst}(I) = \texttt{saturate} ( \texttt{src1}(I) - \texttt{src2} ) \quad \texttt{if mask}(I) \ne0\)
  + Difference between a scalar and an array, when src1 is constructed from Scalar or has the same number of elements as src2.channels(): \(\texttt{dst}(I) = \texttt{saturate} ( \texttt{src1} - \texttt{src2}(I) ) \quad \texttt{if mask}(I) \ne0\)
  + The reverse difference between a scalar and an array in the case of SubRS: \(\texttt{dst}(I) = \texttt{saturate} ( \texttt{src2} - \texttt{src1}(I) ) \quad \texttt{if mask}(I) \ne0\) where I is a multi-dimensional index of array elements. In case of multi-channel arrays, each channel is processed independently.

The first function in the list above can be replaced with matrix expressions: dst = src1 - src2; dst -= src1; // equivalent to subtract(dst, src1, dst); The input arrays and the output array can all have the same or different depths. For example, you can subtract to 8-bit unsigned arrays and store the difference in a 16-bit signed array. Depth of the output array is determined by dtype parameter. In the second and third cases above, as well as in the first case, when src1.depth() == src2.depth(), dtype can be set to the default -1. In this case the output array will have the same depth as the input array, be it src1, src2 or both. **Note:** Saturation is not applied when the output array has the depth CV\_32S. You may even get result of an incorrect sign in the case of overflow.Parameters:src1 - first input array or a scalar.src2 - second input array or a scalar.dst - output array of the same size and the same number of channels as the input array.mask - optional operation mask; this is an 8-bit single channel array that specifies elements of the output array to be changed.dtype - optional depth of the output array SEE: add, addWeighted, scaleAdd, Mat::convertTo

#### subtract public static void subtract([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst)

#### subtract public static void subtract([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask)

#### subtract public static void subtract([Mat](http://docs.google.com/org/opencv/core/Mat.html) src1, [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) src2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask, int dtype)

#### sumElems public static [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) sumElems([Mat](http://docs.google.com/org/opencv/core/Mat.html) src) Calculates the sum of array elements. The function cv::sum calculates and returns the sum of array elements, independently for each channel.Parameters:src - input array that must have from 1 to 4 channels. SEE: countNonZero, mean, meanStdDev, norm, minMaxLoc, reduce Returns:automatically generated

#### SVBackSubst public static void SVBackSubst([Mat](http://docs.google.com/org/opencv/core/Mat.html) w, [Mat](http://docs.google.com/org/opencv/core/Mat.html) u, [Mat](http://docs.google.com/org/opencv/core/Mat.html) vt, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rhs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) wrap SVD::backSubstParameters:w - automatically generatedu - automatically generatedvt - automatically generatedrhs - automatically generateddst - automatically generated

#### SVDecomp public static void SVDecomp([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) w, [Mat](http://docs.google.com/org/opencv/core/Mat.html) u, [Mat](http://docs.google.com/org/opencv/core/Mat.html) vt) wrap SVD::computeParameters:src - automatically generatedw - automatically generatedu - automatically generatedvt - automatically generated

#### SVDecomp public static void SVDecomp([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) w, [Mat](http://docs.google.com/org/opencv/core/Mat.html) u, [Mat](http://docs.google.com/org/opencv/core/Mat.html) vt, int flags) wrap SVD::computeParameters:src - automatically generatedw - automatically generatedu - automatically generatedvt - automatically generatedflags - automatically generated

#### trace public static [Scalar](http://docs.google.com/org/opencv/core/Scalar.html) trace([Mat](http://docs.google.com/org/opencv/core/Mat.html) mtx) Returns the trace of a matrix. The function cv::trace returns the sum of the diagonal elements of the matrix mtx . \(\mathrm{tr} ( \texttt{mtx} ) = \sum \_i \texttt{mtx} (i,i)\)Parameters:mtx - input matrix. Returns:automatically generated

#### transform public static void transform([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) m) Performs the matrix transformation of every array element. The function cv::transform performs the matrix transformation of every element of the array src and stores the results in dst : \(\texttt{dst} (I) = \texttt{m} \cdot \texttt{src} (I)\) (when m.cols=src.channels() ), or \(\texttt{dst} (I) = \texttt{m} \cdot [ \texttt{src} (I); 1]\) (when m.cols=src.channels()+1 ) Every element of the N -channel array src is interpreted as N -element vector that is transformed using the M x N or M x (N+1) matrix m to M-element vector - the corresponding element of the output array dst . The function may be used for geometrical transformation of N -dimensional points, arbitrary linear color space transformation (such as various kinds of RGB to YUV transforms), shuffling the image channels, and so forth.Parameters:src - input array that must have as many channels (1 to 4) as m.cols or m.cols-1.dst - output array of the same size and depth as src; it has as many channels as m.rows.m - transformation 2x2 or 2x3 floating-point matrix. SEE: perspectiveTransform, getAffineTransform, estimateAffine2D, warpAffine, warpPerspective

#### transpose public static void transpose([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Transposes a matrix. The function cv::transpose transposes the matrix src : \(\texttt{dst} (i,j) = \texttt{src} (j,i)\) **Note:** No complex conjugation is done in case of a complex matrix. It should be done separately if needed.Parameters:src - input array.dst - output array of the same type as src.

#### useIPP\_NE public static boolean useIPP\_NE()

#### useIPP\_NotExact public static boolean useIPP\_NotExact()

#### useIPP public static boolean useIPP() proxy for hal::CholeskyReturns:automatically generated

#### vconcat public static void vconcat(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) std::vector<cv::Mat> matrices = { cv::Mat(1, 4, CV\_8UC1, cv::Scalar(1)), cv::Mat(1, 4, CV\_8UC1, cv::Scalar(2)), cv::Mat(1, 4, CV\_8UC1, cv::Scalar(3)),}; cv::Mat out; cv::vconcat( matrices, out ); //out: //[1, 1, 1, 1; // 2, 2, 2, 2; // 3, 3, 3, 3]Parameters:src - input array or vector of matrices. all of the matrices must have the same number of cols and the same depthdst - output array. It has the same number of cols and depth as the src, and the sum of rows of the src. same depth.

* [Overview](http://docs.google.com/overview-summary.html)
* [Package](http://docs.google.com/package-summary.html)
* Class
* [Tree](http://docs.google.com/package-tree.html)
* [Index](http://docs.google.com/index-all.html)
* [Help](http://docs.google.com/help-doc.html)
* [Prev Class](http://docs.google.com/org/opencv/core/Algorithm.html)
* [Next Class](http://docs.google.com/org/opencv/core/Core.MinMaxLocResult.html)
* [Frames](http://docs.google.com/index.html?org/opencv/core/Core.html)
* [No Frames](http://docs.google.com/Core.html)
* [All Classes](http://docs.google.com/allclasses-noframe.html)
* Summary:
* [Nested](#3znysh7) |
* [Field](#2et92p0) |
* [Constr](#tyjcwt) |
* [Method](#3dy6vkm)
* Detail:
* [Field](#4d34og8) |
* [Constr](#3abhhcj) |
* [Method](#49gfa85)

Generated on 2021-04-02 03:15:03 / OpenCV 3.4.14