MetalPerformanceShaders.framework

MetalPerformanceShaders-82

Generated by Doxygen 1.8.13

Contents

Chapter 1

Metal Shaders - High Performance Kernels on Metal

1.1 Introduction

MetalPerformanceShaders.framework is a framework of highly optimized compute and graphics shaders that are designed to integrate easily and efficiently into your Metal application. These data-parallel primitives are specially tuned to take advantage of the unique hardware characteristics of each iOS GPU to ensure optimal performance. Applications adopting MetalPerformanceShaders can be sure of achieving optimal performance without needing to update their own hand-written shaders for each new iOS GPU generation. MetalPerformanceShaders can be used along with the application's existing Metal resources (such as the MTLCommandBuffer, MTLBuffer and MTLTexture objects) and shaders.

In iOS 9, MetalPerformanceShaders.framework provides a series of commonly-used image processing primitives for image effects on Metal textures.

In iOS 10, MetalPerformanceShaders.framework adds support for the following kernels:

- collection of kernels to implement and run neural networks using previously obtained training data, on the GPU
- new image processing filters to perform color-conversion and for building a gaussian pyramid

1.1.1 Using MPS

To use MPS:

```
link:     -framework MetalPerformanceShaders
include:  #include <MetalPerformanceShaders/MetalPerformanceShaders.h>

Advisory: MetalPerformanceShaders features are broken up into many subheaders which are
          included by MetalPerformanceShaders.h.   The exact placement of interfaces in
          headers is subject to change, as functionality in component sub-frameworks can
          move into MPSCore.framework when the functionality needs to be shared by
          multiple components when features are added. To avoid source level breakage,
          #include the top level MetalPerformanceShaders.h header instead of lower
          level headers.  iOS 11 already broke source compatibility for lower level headers
          and future releases will probably do so again. The only supported method of
          including MPS symbols is the top level framework header.
```

On macOS, MetalPerformanceShaders.framework is 64-bit only. If you are still supporting the 32-bit i386 architecture, you can link just your 64-bit slice to MPS using a Xcode user defined build setting. For example, you can add a setting called LINK_MPS:

```
LINK_MPS
    Debug    -framework MetalPerformanceShaders
        Intel architecture            <leave this part empty>
    Release  -framework MetalPerformanceShaders
        Intel architecture            <leave this part empty>
```

The 64-bit intel architectures will inherit from the generic definition on the Debug and Release lines. Next, add to the Other Linker Flags line in your Xcode build settings.

In code segments built for both i386 and x86-64 you will need to keep the i386 segment from attempting to use MPS. In C, C++ and Objective C, a simple #ifdef will work fine.

```
BOOL IsMPSSupported( id <MTLDevice> device )
{
#ifdef __i386__
    return NO;
#else
    return MPSSupportsMTLDevice(device);
#endif
}
```

1.2 Data containers

1.2.1 MTLTextures and MTLBuffers

Most data operated on by Metal Performance Shaders must be in a portable data container appropriate for use on the GPU, such as a MTLTexture, MTLBuffer or MPSImage. The first two should be self-explanatory based on your previous experience with Metal.framework. MPS will use these directly when it can.

Most MPSImage and MPSCNN filters operate only on floating-point or normalized texture formats. If your data is in a UInteger or Integer MTLPixelFormat (e.g. MTLPixelFormatR8Uint as opposed to MTLPixelFormatR8Unorm) then you may need to make a texture view of the texture to change the type using [(id <MTLTexture>) newTextureView↩ WithPixelFormat:(MTLPixelFormat)pixelFormat], to reinterpret the data to a normalized format of corresponding signedness and precision. In certain cases such as thresholding corresponding adjustments (e.g. /255) may have to also be made to parameters passed to the MPSKernel.

1.2.2 MPSImages

Convolutional neural networking (CNN) filters may need more than the four data channels that a MTLTexture can provide. In these cases, the MPSImage is used instead as an abstraction layer on top of a MTLTexture. When more than 4 channels are needed, additional textures in the texture2d array are added to hold additional channels in sets of four. The MPSImage tracks this information as the number of "feature channels" in an image.

1.2.3 MPSTemporaryImages

The MPSTemporaryImage (subclass of MPSImage) extends the MPSImage to provide advanced caching of unused memory to increase performance and reduce memory footprint. They are intended as fast GPU-only storage for intermediate image data needed only transiently within a single MTLCommandBuffer. They accelerate the common case of image data which is created only to be consumed and destroyed immediately by the next operation(s) in a MTLCommandBuffer. MPSTemporaryImages provide convenient and simple way to save memory by automatically aliasing other MPSTemporaryImages in the MTLCommandBuffer. Because they alias (share texel storage with) other textures in the same MTLCommandBuffer, the valid lifetime of the data in a MPSTeporaryImage is extremely short, limited to a portion of a MTLCommandBuffer. You can not read or write data to a MPSTemporaryImage using the CPU, or use the data in other MTLCommandBuffers. Use regular MPSImages for more persistent storage.

1.3 The MPSKernel

The MPSKernel is the base class for all MPS kernels. It defines baseline behavior for all MPS kernels, declaring the device to run the kernel on, some debugging options and a user-friendly label, should one be required. From this are derived the MPSUnaryImageKernel and MPSBinaryImageKernel sub-classes which define shared behavior for most image processing kernels (filters) such as edging modes, clipping and tiling support for image operations that consume one or two source textures. Neither these or the MPSKernel are typically be used directly. They just provide API abstraction and in some cases may allow some level of polymorphic manipulation of MPS image kernel objects.

Subclasses of the MPSUnaryImageKernel and MPSBinaryImageKernel provide specialized -init and -encode methods to encode various image processing primitives into your MTLCommandBuffer, and may also provide additional configurable properties on their own. Many such image filters are available: There are convolutions (generic, box, Sobel, and Gaussian) to do edge detection, sharpening and blurring, morphological operators – Min, Max, Dilate and Erode – and histogram operations. In addition, there are median, resampling filters and others. All of these run on the GPU directly on MTLTextures and MTLBuffers.

As the MPSKernel/MPSUnaryImageKernel/MPSBinaryImageKernel classes serve to unify a diversity of image operations into a simple consistent interface and calling sequence to apply image filters, subclasses implement details that diverge from the norm. For example, some filters may take a small set of parameters (e.g. a convolution kernel) to govern how they function. However, the overall sequence for using MPSKernel subclasses remains the same:

1. Allocate the usual Metal objects: MTLDevice, MTLCommandQueue, and MTLCommandBuffer to drive a Metal compute pipeline. If your application already uses Metal, chances are you have most of these things already. MPS will fit right in to this workflow. It can encode onto MTLCommandBuffers inline with your own workload.

2. Create an appropriate MPSKernel object. For example, if you want to do a Gaussian blur, make a MPS←ImageGaussianBlur object. MPSKernel objects are generally light weight but can be reused to save some setup time. They can not be used by multiple threads concurrently, so if you are using Metal from many threads concurrently, make extra MPSKernel objects. MPSKernel objects conform to <NSCopying>.

3. Call [MPSKernelSubclass encodeToCommandBuffer:...]. Parameters for other -encode... calls vary by filter type, but operate similarly. They create a MTLCommandEncoder, write commands to run the filter into the MTLCommandBuffer and then end the MTLCommandEncoder. This means you must call -endEncoding on your current MTLCommandEncoder before calling a MPSKernel encode method. You can at this point release the MPSKernel or keep it around to use again to save some setup cost.

4. If you wish to encode futher commands of your own on the MTLCommandBuffer, you must create a new MTLCommandEncoder to do so.

5. (Standard Metal) When you are done with the MTLCommandBuffer, submit it to the device using typical Metal commands, such as [MTLCommandBuffer commit]. The MPS filter will begin running on the GPU. You can either use [MTLCommandBuffer waitUntilCompleted] or [MTLCommandBuffer addCompletedHandler:] to be notified when the work is done.

Each MPSKernel is allocated against a particular MTLDevice. A single filter may not be used with multiple M←TLDevices. (You will need to make multiple MPSKernels for that.) This is necessary because the [MPSKernel initWithDevice:...] methods sometimes allocate MTLBuffers and MTLTextures to hold data passed in as parameters to the -init method and a MTLDevice is required to make those. MPSKernels provide a copy method that allow them to be copied for a new device.

MPSKernel objects are not entirely thread safe. While they may be used in a multithreaded context, you should not attempt to have multiple MPSKernel objects writing to the same MTLCommandBuffer at the same time. They share restrictions with the MTLCommandEncoder in this regard. In limited circumstances, the same MPSKernel can be used to write to multiple MTLCommandBuffers concurrently. However, that only works if the MPSKernel is treated as an immutable object. That is, if MPSKernel subclass properties of a shared filter are changed, then the change can be reflected on the other thread while the other thread is encoding its work, leading to undefined behavior. It is generally safest to just make copies of MPSKernel objects, one for each thread.

For more information, please see MPSTypes.h.

1.3.1 MPS{Unary/Binary}ImageKernel properties

The MPS{Unary/Binary}ImageKernel base classes define several properties common to all MPSKernels:

1.3.1.1 MPSKernel clipRect

The clipRect property, common to MPSKernel sublcasses that write to a destination texture, describes the sub-rectangle of the destination texture overwritten by the filter. If the clipRect is larger than the destination texture, the intersection between the clipRect and destination texture bounds will be used. The clipRect may be used to avoid doing work to obscured regions of the destination image, or to manage tiling and to limit operations to parts of an image if for example, the user drew a rectangle on the screen and asked you to just apply the filter there.

```
extern MTLRegion MPSRectNoClip; //Pass this rectangle to fill the entire destination texture.
```

1.3.1.2 MPSOffset

The offset (or primaryOffset or secondaryOffset) property, common to MPSKernel subclasses that use a source texture from which pixel data is read, describes the positioning of the source image relative to the result texture. A offset of {0,0,0} indicates that the top left pixel of the source texture is the center pixel used to create the top left corner of the destination texture clipRect. An offset of {1,2,0} positions the top left corner of the clipRect at {x=1, y=2, z=0} of the source image. The offset is the position of the top left corner of the clipRect in the source coordinate frame. It can be used for tiling and for translating an image up/down or left/right by pixel increments. If there is no clipRect then the offset is the top left corner of the region read by the filter. If there are multiple source textures, then the primaryOffset describes the top left corner of the region read in the primary source texture. The secondaryOffset describes the top left corner of the region read in the secondary source texture, and so forth.

1.3.1.3 MPSKernelEdgeMode

The edgeMode (or primaryEdgeMode or secondaryEdgeMode)describes the behavior of texture reads that stray off the edge of the source image. This can happen if the offset is negative, meaning read off the top or left edge of the image. It can also happen if the clipRect.size + offset is larger than the source image, meaning read off the bottom and right of the image. It is also possible for filters that have a filter window that stretches to examine neighboring pixels, such as convolution, morphology and resampling filters. If there are multiple source textures, then the primaryEdgeMode describes the MPSKernelEdgeMode to use with primary source texture. The secondaryEdge↩Mode describes the MPSKernelEdgeMode to use with the secondary source texture, and so forth.

```
typedef NS_ENUM(NSUInteger, MPSImageEdgeMode)

MPSImageEdgeModeZero        Out of bound pixels are (0,0,0,1) for image formats without
                                 alpha channel and (0,0,0,0) for image with pixel format with an
                                 alpha channel

MPSImageEdgeModeClamp       Out of bound pixels are clamped to nearest edge pixel
```

1.3.1.4 MPSKernelOptions

Each MPSKernel takes a MPSKernelOptions bit mask to indicate various options to use when running the filter:

```
typedef NS_OPTIONS(NSUInteger, MPSKernelOptions)

MPSKernelOptionsNone                        Use default options

MPSKernelOptionsSkipAPIValidation           Do not spend time looking at parameters passed to MPS
                                           for errors.

MPSKernelOptionsAllowReducedPrecision       When possible, MPSKernels use a higher precision data representation
                                           the destination storage format to avoid excessive accumulation of comp
                                           rounding error in the result. MPSKernelOptionsAllowReducedPrecision ac
                                           MPSKernel that the destination storage format already has too much pre
                                           what is ultimately required downstream, and the MPSKernel may use redu
                                           internally when it feels that a less precise result would yield better
                                           When enabled, the precision of the result may vary by hardware and ope
```

1.4 Available MPSKernels

1.4.1 Image Convolution

1.4.1.1 The Image Convolution Kernel

The convolution filter is at its simplest the weighted average of a pixel with its nearest neighbors. The weights are provided by a convolution kernel. The number and position of the nearest neighbors that are considered are given by the size of the convolution kernel. For example, a convolution kernel might be the following 5x5 array of weights:

```
1       2       3       2       1
2       4       6       4       2
3       6      [9]      6       3
2       4       6       4       2
1       2       3       2       1
```

In order to calculate this 5x5 convolution result, one would multiply all of the pixels in the image within (5/2=) 2 pixels of the desired pixel by its corresponding weight, add them up and divide by a divisor to renormalize the results. Then, repeat for all other pixels in the area you wish to convolve.

For those MPS filters where the convolution kernel is passed in, you provide the kernel as a normalized float array. That is, the kernel weights have the divisor already divided into them and as a consequence should usually sum to 1.0. In our tent example above, the sum over the area of the kernel is 81, so one would normalize it as follows:

```
1/81    2/81    3/81    2/81    1/81
2/81    4/81    6/81    4/81    2/81
3/81    6/81   [9/81]   6/81    3/81
2/81    4/81    6/81    4/81    2/81
1/81    2/81    3/81    2/81    1/81
```

It is not strictly necessary that the filter weights add up to 1.0f. Edge detection filters frequently add up to zero. You may decide to have the area under the filter be a bit bigger or smaller than 1.0 to increase or reduce the contrast in the result.

The MxN kernel is passed in as a 1-dimensional data array in row major order.

Some convolution filters also have a notion of a bias. This is a number to be added to the result before it is written out to result texture. A bias might be used to uniformly brighten an image, set a video range baseline (e.g. 0 might actually be encoded as 16/255) or to make negative signal representable on a unorm image.

```
A unorm image is an image comprised of unsigned normalized samples. A typical 8-bit image (e.g.
MTLPixelFormatRGBA8Unorm) is a unorm image. It has unsigned samples that represent values between
[0,1]. In the case of MTLPixelFormatRGBA8Unorm, the encoding of 0 is 0, and the encoding of 1.0f
is UINT8_MAX (255).
```

1.4.1.2 The Box, Tent and Gaussian Filters

There are many different convolution kernel shapes which can produce different results. A kernel consisting of all 1's is called a Box filter. It is very quick to calculate and may run nearly as fast as a texture copy, even for very large blur radii. The blur effect that you get, however, can be square in appearance and may not be entirely appealing under close scrutiny. A second pass of the box will lead to a Tent kernel. (The 5x5 tent above can be reduced into two 3x3 Box filters.) Its appearance is more pleasing. Tent operations can be found in sample code for window shadows. Both Box and Tent filters are provided by MPS. Multiple passes of a box and/or tent filters will tend to converge towards a gaussian line shape and produce a smoother blur effect. MPS also provides a Gaussian blur, though it uses a different method.

1.4.1.3 Laplacian and Unsharp Mask Filters

One can in practice also subtract a blur from the image to produce a sharpening effect (unsharp mask). This is done by preparing a convolution kernel which is a scaled image less a blur to reduce the low frequency component of an image. This can reduce blur, but may also emphasize noise in the image. As an example, we can do identity minus a box blur:

```
             | 1   1   1 |
k0 * [1] -   | 1   1   1 | * k2
             | 1   1   1 |
```

If we pick k0 = 9 and k2 = 1, so that the two kernels have equal weight, we arrive at:

```
|-1  -1  -1 |
|-1   8  -1 |
|-1  -1  -1 |
```

This is a Laplacian filter for calculating image gradients (including diagonals in this case).

```
Caution: because this convolution kernel has negative regions, it can produce negative
results as well as positive ones from ordinary image data. If you intend to store the
result in a unorm texture, you'll need to scale it and add a positive value to it to avoid
having the negative signal clamped off. (e.g. p' = 0.5*p+0.5).
```

An unsharp mask filter is the sum between the original image and a scaled result of the Laplacian filter. The scaling factor (and filter size and shape) adjusts the nature of the low frequency signal and the degree to which it is removed. This work can usually be combined into the convolution kernel, to do the whole thing in one pass.

1.4.1.4 Sobel Edge detection

Instead of integrating over an area, Convolution filters can also differentiate over an area by subtracting adjacent pixels. One such filter is the Sobel edge detection filter. It produces bright signal where there are large differences between one pixel and the next and black elsewhere:

```
   | -1  0   1 |       | 1   2   1 |
Gx=| -2  0   2 |   Gy= | 0   0   0 |
   | -1  0   1 |       |-1  -2  -1 |


   result = sqrt( Convolve(src, Gx) * Convolve(src * Gx) +
                  Convolve(src, Gy) * Convolve(src * Gy))
```

1.4.1.5 Other Filters

Other effects can be achieved as well, such as emboss:

```
-1   0   0
 0   0   0
 0   0   1
```

1.4.1.6 Separable Convolution

Some convolution kernels are separable. That is, the filter weights can be factored into the product of two smaller sets of weights. As an example, the tent kernel shown above can be factored into a horizontal and vertical 1-dimensional kernel each containing [1 2 3 2 1]. In this way, what might otherwise have been a 5x5 convolution with 25 multiplies and 24 adds is instead a 5x1 and 1x5 convolution with a total of 10 multiplies and 8 adds and possibly some extra load/store traffic for the two-pass algorithm. The savings get bigger for bigger filter areas. MPS convolution filters will automatically separate kernels to run faster, when possible. Some filters with fixed kernels such as Box and Guassian are inherently separable. We attempt to factor the general convolution kernel into 2 1D kernels in the -initWithDevice:... method. If you want to factor it yourself, make two MPSImageConvolution objects with 1D kernels.

1.4.1.7 Convolutions in MPS

Convolution filters provided by MPS include:

```
MPSImageConvolution        <MPSImage/MPSImageConvolution.h>         General convolution
MPSImageGassianBlur        <MPSImage/MPSImageConvolution.h>         Gaussian blur
MPSImageBox                <MPSImage/MPSImageConvolution.h>         Box blur
MPSImageTent               <MPSImage/MPSImageConvolution.h>         Tent blur
```

1.4.2 Morphology

Morphological operators are similar to convolutions in that they find a result by looking at the nearest neighbors of each pixel in the image. Instead of calculating a weighted average, morphological operators scan the kernel area looking for the maximum or minimum pixel value. The MPSImageAreaMax and MPSImageAreaMin filters return the raw maximum and minimum color channel value in the kernel area for each pixel, respectively. The MPSImageDilate and MPSImageErode filters do the same thing, except that the probe shape need not be a rectangle, and instead can be nearly any shape you desire, such as a antialiased oval, star or heart shape.

When applied, the max and dilate filters have the effect of adding their shape on to the periphery of bright objects in the image. A single bright pixel, such as might be found in a photograph of a starry night sky will become the shape of a probe – a rectangle for max, and perhaps a 5-pointed star if that is the shape you chose for the dilate filter kernel. Larger objects will adopt more rectangular or star quality into their shape. (An oval or circular probe would round the corners of a rectangular object, for example.) The min and erode filters do similar things to the darker regions of the image.

When a dilate filter is followed by an erode filter (or max followed by min) with similar filters, the effect is known as a close operator. Expanding bright areas only to erode them away again leaves most of the image in roughly the same shape as it started, but small dark areas that are completely removed by the dilate operator are not replaced by the erode. Dark noise may be removed. Small enclosed dark area may be completely filled in by bright signal. Similarly erode followed by dilate is an open operator. It will tend to remove bright fine detail and fill in small bright areas surrounded by dark lines.

To make a MPS morphology filter with a text glyph draw black text on a white background. MPS morphology filters must have a center pixel with value 0.

Morphology filters provided by MPS include:

```
MPSImageAreaMax   <MPSImage/MPSImageMorphology.h>       Area Max
MPSImageAreaMin   <MPSImage/MPSImageMorphology.h>       Area Min
MPSImageDilate    <MPSImage/MPSImageMorphology.h>       Dilate
MPSImageErode     <MPSImage/MPSImageMorphology.h>       Erode
```

1.4.3 Histogram

A image may be examined by taking the histogram of its pixels. This gives the distribution of the various intensities per color channel. The MPSImageHistogram filter can be used to calculate a histogram for a MTLTexture.

In some cases, as a result of image processing operations the very dark and light regions of the intensity spectrum can become unpopulated. Perhaps a photograph is underexposed or overexposed. The MPSImageHistogramEqualization filter will redistribute the intensities to a more uniform distribution, correcting such problems. The MPSImageHistogramSpecification class allows you to cause an image to conform to a different histogram.

Histogram filters provided by MPS include:

```
MPSImageHistogram               <MPSImage/MPSImageHistogram.h>     Calculate the histogram of an image
MPSImageHistogramEqualization   <MPSImage/MPSImageHistogram.h>     Redistribute intensity in an image to equali
                                                                     the histogram

MPSImageHistogramSpecification <MPSImage/MPSImageHistogram.h>     A generalized version of histogram equalizat
                                                                     operation. Convert the image so that its h
                                                                     matches the desired histogram provided to
                                                                     histogram specification filter.
```

1.4.4 Image Median

Median filters find the median value in a region surrounding each pixel in the source image. It is frequently used to remove noise from the image, but may also be used to remove fine detail like a open filter. It is widely used in image processing because in many cases it can remove noise while at the same time preserving edges.

Median filters provided by MPS include:

```
MPSImageMedian                 <MPSImage/MPSImageMedian.h>          Calculate the median of an image using a
                                                                 square filter window.
```

1.4.5 Image Resampling

Resampling operations are used to convert one regular array of pixels to another regular array of pixels, typically along a different set of axes and/or using a different sampling period. Changing the sampling period will enlarge or reduce images and/or distort the aspect ratio. Change of axis results in rotations or arbitrary affine transforms.

For most imaging work on the GPU, resampling can be quickly and simply done as part of another pass using a Euler matrices or quaternions to transform the coordinate space followed by linear filtering to interpolate the value found there. However, this can lead to somewhat muddy images and may result in loss of signal when downsampling by more than a factor of two unless a low pass filter is applied first. It is also prone to the development of Moire patterns in regions of the image with regularly repeating signal, such as a picture of a masonry grid on the side of a building.

The MPS resampling routines use a higher quality (but more expensive) Lanczos resampling algorithm. Especially with photographic images, it will usually produce a much nicer result. It does not require a low pass filter be applied to the image before down sampling. However, some ringing can occur near high frequency regions of the image, making the algorithm less suitable for vector art.

MetalPerformanceShaders.framework provides a MPSImageLanczosScale function to allow for simple resizing of images into the clipRect of the result image. It can operate with preservation of aspect ratio or not.

```
MPSImageLanczosScale              <MPSImage/MPSResample.h>   Resize or adjust aspect ratio of an image.
```

1.4.6 Image Threshold

Thresholding operations are commonly used to separate elements of image structure from the rest of an image. Generally, these operate by making some sort of simple comparison test, for example color_intensity $>$ 0.5, and then writing out 0 or 1 (actual values configurable) depending on the truth or falsehood of the result. It is frequently used in computer vision, or to accentuate edge detection filters.

A variety of thresholding operators are supported:

```
MPSImageThresholdBinary           <MPSImage/MPSImageThreshold.h>  srcPixel > thresholdVal ? maxVal : 0
MPSImageThresholdBinaryInverse    <MPSImage/MPSImageThreshold.h>  srcPixel > thresholdVal ? 0 : maxVal
MPSImageThresholdTruncate         <MPSImage/MPSImageThreshold.h>  srcPixel > thresholdVal ? thresholdVal : src
MPSImageThresholdToZero           <MPSImage/MPSImageThreshold.h>  srcPixel > thresholdVal ? srcPixel : 0
MPSImageThresholdToZeroInverse    <MPSImage/MPSImageThreshold.h>  srcPixel > thresholdVal ? 0 : srcPixel
```

1.4.7 Math Filters

Arithmetic filters take two source images, a primary source image and a secondary source image, as input and output a single destination image. The filters apply an element-wise arithmetic operator to each pixel in a primary source image and a corresponding pixel in a secondary source image over a specified region. The supported arithmetic operators are addition, subtraction, multiplication, and division.

These filters take additional parameters: primaryScale, secondaryScale, and bias and apply them to the primary source pixel (x) and secondary source pixel (y) in the following way:

```
MPSImageAdd          <MPSImage/MPSImageMath.h>  Element-wise addition operator:      result = ((primaryScale *
MPSImageSubtract     <MPSImage/MPSImageMath.h>  Element-wise subtraction operator    result = ((primaryScale *
MPSImageMultiply     <MPSImage/MPSImageMath.h>  Element-wise multiplication operator result = ((primaryScale *
MPSImageDivide       <MPSImage/MPSImageMath.h>  Element-wise division operator       result = ((primaryScale *
```

These filters also take the following additional parameters: secondarySourceStrideInPixelsX and secondary↩ SourceStrideInPixelsY. The default value of these parameters is 1. Setting both of these parameters to 0 results in the secondarySource image being handled as a single pixel.

1.4.8 Convolutional Neural Networks

Convolutional Neural Networks (CNN) is a machine learning technique that attempts to model the visual cortex as a sequence of convolution, rectification, pooling and normalization steps. Several CNN filters commonly derived from the MPSCNNKernel base class are provided to help you implement these steps as efficiently as possible.

```
MPSCNNNeuronLinear                <MPSNeuralNetwork/MPSCNNConvolution.h>     A linear neuron activation funct
MPSCNNNeuronReLU                  <MPSNeuralNetwork/MPSCNNConvolution.h>     A neuron activation function wit
MPSCNNNeuronSigmoid               <MPSNeuralNetwork/MPSCNNConvolution.h>     A sigmoid neuron activation func
MPSCNNNeuronHardSigmoid           <MPSNeuralNetwork/MPSCNNConvolution.h>     A hard sigmoid neuron activation
MPSCNNNeuronTanH                  <MPSNeuralNetwork/MPSCNNConvolution.h>     A neuron activation function usi
MPSCNNNeuronAbsolute              <MPSNeuralNetwork/MPSCNNConvolution.h>     An absolute neuron activation fu
MPSCNNNeuronSoftPlus              <MPSNeuralNetwork/MPSCNNConvolution.h>     A parametric SoftPlus neuron act
MPSCNNNeuronSoftSign              <MPSNeuralNetwork/MPSCNNConvolution.h>     A SoftSign neuron activation fun
MPSCNNNeuronELU                   <MPSNeuralNetwork/MPSCNNConvolution.h>     A parametric ELU neuron activati
MPSCNNConvolution                 <MPSNeuralNetwork/MPSCNNConvolution.h>     A 4D convolution tensor
MPSCNNFullyConnected              <MPSNeuralNetwork/MPSCNNConvolution.h>     A fully connected CNN layer
MPSCNNPoolingMax                  <MPSNeuralNetwork/MPSCNNPooling.h>         The maximum value in the pooling
MPSCNNPoolingAverage              <MPSNeuralNetwork/MPSCNNPooling.h>         The average value in the pooling
MPSCNNPoolingL2Norm               <MPSNeuralNetwork/MPSCNNPooling.h>         The L2-Norm value in the pooling
MPSCNNDilatedPoolingMax           <MPSNeuralNetwork/MPSCNNPooling.h>         The maximum value in the dilated
MPSCNNSpatialNormalization        <MPSNeuralNetwork/MPSCNNNormalization.h>
MPSCNNCrossChannelNormalization   <MPSNeuralNetwork/MPSCNNNormalization.h>
MPSCNNSoftmax                     <MPSNeuralNetwork/MPSCNNSoftMax.h>         exp(pixel(x,y,k))/sum(exp(pixel(
MPSCNNLogSoftmax                  <MPSNeuralNetwork/MPSCNNSoftMax.h>         pixel(x,y,k) - ln(sum(exp(pixel(
MPSCNNUpsamplingNearest           <MPSNeuralNetwork/MPSCNNUpsampling.h>      A nearest upsampling layer.
MPSCNNUpsamplingBilinear          <MPSNeuralNetwork/MPSCNNUpsampling.h>      A bilinear upsampling layer.
```

MPSCNNKernels operate on MPSImages. MPSImages are at their core MTLTextures. However, whereas MTL←Textures commonly represent image or texel data, a MPSImage is a more abstract representation of image features. The channels within a MPSImage do not necessarily correspond to colors in a color space. (Though, they can.) As a result, there can be many more than four of them. 32 or 64 channels per pixel is not uncommon. This is achieved on the MTLTexture hardware abstraction by inserting extra RGBA pixels to handle the additional feature channels (if any) beyond 4. These extra pixels are stored as multiple slices of a 2D image array. Thus, each CNN pixel in a 32-channel image is represented as 8 array slices, with 4-channels stored per-pixel in each slice. The width and height of the MTLTexture is the same as the width and height of the MPSImage. The number of slices in the MTLTexture is given by the number of feature channels rounded up to a multiple of 4.

MPSImages can be created from existing MTLTextures. They may also be created anew from a MPSImage←Descriptor and backed with either standard texture memory, or as MPSTemporaryImages using memory drawn from MPS's internal cached texture backing store. MPSTemporaryImages can provide great memory usage and CPU time savings, but come with significant restrictions that should be understood before using them. For example, their contents are only valid during the GPU-side execution of a single MTLCommandBuffer and can not be read from or written to by the CPU. They are provided as an efficient way to hold CNN computations that are used immediately within the scope of the same MTLCommandBuffer and then discarded. We also support concatenation by allowing the user to define from which destination feature channel to start writing the output of the current layer. In this way the application can make a large MPSImage or MPSTemporaryImage and fill in parts of it with multiple layers (as long as the destination feature channel offset is a multiple of 4).

Some CNN Tips:

- Think carefully about the edge mode requested for pooling layers. The default is clamp to zero, but there are times when clamp to edge value may be better.
- To avoid falling off the edge of an image for filters that have a filter area (convolution, pooling) set the MPS←CNNKernel.offset = (MPSOffset){ .x = kernelWidth/2, .y = kernelHeight/2, .z = 0}; and reduce the size of the output image by {kernelWidth-1, kernelHeight-1,0}. The filter area stretcheds up and to the left of the MP←SCNNKernel.offset by {kernelWidth/2, kernelHeight/2}. While consistent with other MPS imaging operations, this behavior is different from some other CNN implementations.
- Please remember: MPSCNNConvolution takes weights in the order weight[outputChannels][kernel←Height][kernelWidth][inputChannels / groups] MPSCNNFullyConnected takes weights in the order weight[outputChannels][sourceWidth][sourceHeight][inputChannels]
- Initialize MPSCNNKernels once and reuse them
- You can use MPSCNNNeurons and other Filters in MPS to perform pre-processing of images, such as scaling and resizing.
- Specify a neuron filter with MPSCNNConvolution descriptor to combine the convolution and neuron operations.
- Use MPSTemporaryImages for intermediate images that live for a short period of time (less than one MTL←CommandBuffer). MPSTemporaryImages can reduce the amount of memory used by the convolutional neural network by several fold, and similarly reduce the amount of CPU time spent allocating storage and latency between MTLCommandBuffer.commit and when the work actually starts on the GPU. MPSTemporaryImage are for short lived storage within the time period of the execution of a single MTLCommandBuffer. You can not read or write to a MPSTemporaryImage using the CPU. Generally, they should be created as needed and thrown away promptly. Persistent objects should not retain them. Please be sure to understand the use of the MPSTemporaryImage.readCount.
- Because MPS encodes its work in place in your MTLCommandBuffer, you always have the option to insert your own code in between MPSCNNKernels as a Metal shader for tasks not covered by MPS. You need not use MPS for everything.

1.5 MPS API validation

MPS uses the same API validation layer that Metal uses to alert you to API mistakes while you are developing your code. While this option is turned on (Xcode: Edit Scheme: options: Metal API Validation), common programming errors will either trigger an assert or send a warning to the the debug log. Except in the case of serious errors, little or no spew should arrive in the console under standard usage. You can also try the MPSKernel.options parameter MPSKernelOptionsSkipAPIValidation to skip most of this checking. The flag may also lead to small reductions in CPU cost.

Note: where APIs are tagged nonnull, MPS expects that the value is not NULL. The validation layer may do some checking and assert. If you turn that off, then undefined behavior is the result of passing nil, and your application will likely be terminated.

1.6 How to Add MetalPerformanceShaders.framework to your project

Xcode:

```
1.  Click project file at left, then appropriate target, then select Build Phases.
2.  Open the "Link Binary With Libraries" disclosure triangle
3.  Click the [+] button in the "Link Binary With Libraries" view to add a new library
4.  Select MetalPerformanceShaders.framework from the list.
5.  Click the Add button.
```

Command Line:

```
clang  -framework MetalPerformanceShaders    file.c -o file.o
```

1.7 How to Determine if MPS Works on Your Device

To test whether MPS works on your device, you may call MPSSupportsMTLDevice(id<MTLDevice>). It will return YES if the device is supported.

1.8 In Place Operation

Some MPS filters can operate in place. In-place operation means that the same texture is used to hold both the input image and the result image. Operating in place is a great way to save memory, time and energy. You can use a MPS filter in place using [MPSKernel encodeToCommandBuffer:inPlaceTexture:copyAllocator:].

Unfortunately, it is not always possible for MPS filters to run in place. Whether a particular MPSKernel can operate in place can vary according to the hardware it is running on, the operating system version and the parameters and properties passed to it. You may not assume that because a MPSKernel works in place today on a particular device that it will continue to do so in the future.

To simplify error handling with failed in-place operation, [MPSKernel encodeToCommandBuffer:inPlaceTexture↵:fallbackCopyAllocator:] takes an optional MPSCopyAllocator parameter. It is used to create a new texture when in-place operation is not possible so as to allow the operation to proceed out of place in a reliable fashion instead. (When this happens the input texture is released and replaced with a new texture.) To make use of this feature, you will need to write a MPSCopyAllocator block.

1.8.1 MPSCopyAllocator

Some MPSKernel objects may not be able to operate in place. When that occurs, and in-place operation is requested, MPS will call back to this block to get a new texture to overwrite instead. To avoid spending long periods of time allocating pages to back the MTLTexture, the block should attempt to reuse textures. The texture returned from the MPSCopyAllocator will be returned instead of the sourceTexture from the MPSKernel method on return. Here is a minimal MPSCopyAllocator implementation:

```
// A MPSCopyAllocator to handle cases where in-place operation fails.
MPSCopyAllocator myAllocator = ^id <MTLTexture>( MPSKernel * __nonnull filter,
                                                __nonnull id <MTLCommandBuffer> cmdBuf,
                                                __nonnull id <MTLTexture> sourceTexture)
{
    MTLPixelFormat format = sourceTexture.pixelFormat;  // FIXME: is this format writable?
    MTLTextureDescriptor *d = [MTLTextureDescriptor texture2DDescriptorWithPixelFormat: format
                                 width: sourceTexture.width
                                height: sourceTexture.height
                             mipmapped: NO];
    d.usage = MTLTextureUsageShaderRead | MTLTextureUsageShaderWrite;

    //FIXME: Allocating a new texture each time is slow. They take up to 1 ms each.
    //       There are not too many milliseconds in a video frame! You can recycle
    //       old textures (or MTLBuffers and make textures from them) and reuse
    //       the memory here.
    id <MTLTexture> result = [cmdBuf.device newTextureWithDescriptor: d];

    // FIXME: If there is any metadata associated with sourceTexture such as colorspace
    //        information, MTLResource.label, MTLResource.cpuCacheMode mode,
    //        MTLResource.MTLPurgeableState, etc., it may need to be similarly associated
    //        with the new texture, to avoid losing your metadata.

    // FIXME: If filter.clipRect doesn't cover the entire image, you may need to copy
    //        pixels from sourceTexture to result or regions of result will be
    //        uninitialized. You can make a MTLCommandEncoder to encode work on the
    //        MTLCommandBuffer here to do that work, if necessary. It will be scheduled
    //        to run immediately before the MPSKernel work. Do not call
    //        [MTLCommandBuffer enqueue/commit/waitUntilCompleted/waitUntilScheduled]
    //        in the MPSCopyAllocator block. Make sure to call -endEncoding on the
    //        MTLCommandEncoder so that the MTLCommandBuffer has no active encoder
    //        before returning.

    // CAUTION: The next command placed on the MTLCommandBuffer after the MPSCopyAllocator
    //          returns is almost assuredly going to be encoded with a MTLComputeCommandEncoder.
    //          Creating any other type of encoder in the MPSCopyAllocator will probably cost
    //          an additional 0.5 ms of both CPU _AND_ GPU time (or more!) due to a double
    //          mode switch penalty.

    return result;
    // d is autoreleased
};
```

```
filter          A valid pointer to the MPSKernel that is calling the MPSCopyAllocator. From
                it you can get the clipRect of the intended operation.
cmdBuf          A valid MTLCommandBuffer. It can be used to obtain the device against
                which to allocate the new texture. You may also enqueue operations on
                the commandBuffer to initialize the texture. You may not submit, enqueue
                or wait for completion of the command buffer.
sourceTexture   The texture that is providing the source image for the filter. You may
                wish to copy its size and MTLPixelFormat for the new texture, but it is
                not requred.

return          A new valid MTLTexture to use as the destination for the MPSKernel.
                The format of the returned texture does not need to match sourceTexture.
```

1.9 The MPSNNGraph

New for macOS 10.13, iOS/tvOS 11 is a higher level graph API, intended to simplify the creation of neural networks. The graph is a network of MPSNNFilterNodes, MPSNNImageNodes and MPSNNStateNodes. MPSNNImageNodes represent MPSImages or MPSTemporaryImages. MPSNNFilterNodes represent MPSCNNKernel objects – each of the lower level MPSCNNKernel subclasses has a sister object that is a subclass of the MPSNNFilterNode. Finally, MPSStateNodes stand in for MPSState objects.

MPSState objects are also new for macOS 10.13, iOS/tvOS 11. They stand in for bits of opaque state that need to be handed between filter nodes. For example, a MPSCNNConvolutionTranspose filter may need to know the original size of the filter passed into the corresponding MPSCNNConvolution node farther up the tree. There is a corresponding MPSCNNConvolutionState object that tracks this information. You will encounter state objects only infrequently. Most graphs are made up of images and filters.

To represent a graph, one usually first creates a MPSNNImageNode. This represents the input image or tensor into the graph. Next one creates a the first filter node to process that input image. For example, we may make a MPSCNNNeuronLinearNode to normalize the data before the rest of the graph sees it. (y = 2x-1) Then, we can add our first convolution in the graph.

```
//  Graph:   [Image] -> [Linear neuron filter] -> (result image) -> [convolution filter] -> (result
       image)...
MPSNNImageNode *startNode = [MPSImageNode nodeWithHandle: nil];
MPSCNNNeuronLinearNode *norm = [MPSCNNNeuronLinearNode
      nodeWithSource: startNode a: 2.0f b: -1.0f ];
MPSCNNConvolutionNode *c = [MPSCNNConvolutionNode nodeWithSource:
       norm.resultImage
                                                        weights: [[MyWeights alloc] initWithPath: "
      weights1.dat"]];
...
```

There are some features to notice about each object. First of all, each image node can have a handle associated with it. The handle is your object that you write. It should conform to the <MPSHandle> protocol, which specifies that the object should have a label and conform to NSSecureCoding. (The MTLTexture does have a label property but doesn't conform to NSSecureCoding.) NSSecureCoding is used when you save the graph to disk using a NS↩ Coder. It isn't used otherwise. You can use a MTLResource here if you don't plan to save the graph to disk. What is the handle for? When the MPSNNGraph object is constructed – the MPSNNGraph takes the network of filter, state and image nodes and rolls it into an object that can actually encode work to a MTLCommandBuffer – the graph object will traverse the graph from back to front and determine which image nodes are not produced by filters in the graph. These, it will intepret to be graph input images. There may be input states too. When it does so, it will represent these image and state nodes as the handles you attach to them. Therefore, the handles probably should be objects of your own making that refer back to your own data structures that identify various images that you know about.

Continuing on to the neuron filter, which we are using to just take the usual image [0,1] range and stretch to [-1,1] before the rest of the graph sees it, we see we can pass in the linear filter parameters when constructing it here. All filter nodes also produce a result image. This is used as the argument when constructing the convolution filter node next, to show that the product of the neuron filter is the input to the convolution filter.

The convolution object constructor also takes a weights object. The weights object is an object that you write that conforms to the MPSCNNConvolutionDataSource protocol. MPS does not provide an object that conforms to this protocol, though you can see some examples in sample code that use this interface. The convolution data source object is designed to provide for deferred loading of convolution weights. Convolution weights can be large. In aggregate, the storage for all the weights in the MPSNNGraph, plus the storage for your copy of them might start to approach the storage limits of the machine for larger graphs. In order to lessen this impact, the convolution weights are unpacked for each convolution in turn and then purged from memory so that only the single MPSNNGraph copy remains. This happens when the MPSCNNConvolutionDataSource load and purge methods are called. You should not load the weights until -load is called. (You probably should however verify that the file, if any, is there and is well formed in the object -init method.) When -purge is called, you should release any bulky storage that the object owns and and make the object as light weight as is reasonable. The MPSCNNConvolutionDataSource.descriptor may include a neuron filter operation.

Other object types should be straightforward.

1.10 MPSNNGraph usage

Once the network of MPSNNFilterNodes, MPSNNImageNodes and MPSNNStateNodes is created, the next step is to identify the MPSNNImageNode that contains the result of your graph – typically, this is the last one you made – and make a MPSNNGraph with it:

```
MPSNNGraph *graph = [[MPSNNGraph alloc] initWithDevice: mtlDevice
                                           resultImage: resultImage];
```

If graph creation fails, nil will be returned here. When it is constructed, the graph iterates over the network of nodes, starting at the result image and working backwards. Any MPSNNImageNodes and states that are used that are not created by a MPSNNFilterNode are interpreted to be graph inputs. The identity of these are given by the MPSNNGraph.sourceImageHandles and MPSNNGraph.sourceStateHandles. Each handle is your object that refers back to a particular image or state node. The order of the handles matches the order of the images or states that should be passed to the [MPSNNGraph encodeToCommandBuffer:...] call. Similarly, you can get the identity of any intermediate images that you requested to see (See MPSNNImageNode.exportFromGraph property) and the identity of any result MPSStates that are produced by the graph that are not used. The graph has a destinationImageAllocator that overrides the MPSNNImageNode.destinationImageAllocator. (see subsection MPSNNGraph intermediate image allocation) Typically, this serves to make a default temporary image into a normal image, as a convenience.

When you are ready to encode a graph to a command buffer, the operation follows as per much of the rest of MPS.

```
id <MTLDevice> mtlDevice = MTLCreateSystemDefaultDevice();
id <MTLCommandQueue> mtlCommandQueue.commandBuffer = mtlDevice.newCommandQueue;
id <MTLCommandBuffer> cmdBuf = mtlCommandQueue.commandBuffer;
MPSImage *inputImage = [[MPSImage alloc] initWithDevice: mtlDevice imageDescriptor:
      myDescriptor];
// put some data into the input image here. See MTLTexture.replaceBytes...
MPSImage * result = [myGraph encodeToCommandBuffer: cmdBuf sourceImages: @[inputImage] ];
[cmdBuf commit];
[cmdBuf waitForCompletion];
```

Obviously, if you have more work to do before or after the graph, it might be better to add it to the command buffer before committing it, rather than paying for an extra synchronization from [id <MTLCommandBuffer> waitForCompletion].

1.11 MPSNNGraph intermediate image sizing and centering

The MPSNNGraph will automatically size and center the intermediate images that appear in the graph. However, different neural network frameworks do so differently. In addition, some filters may at times operate on only valid pixels in the source image, whereas others may "look beyond the edges" so as to keep the result image size the same as the input. Occasionally some filters will want to produce results for which any input is valid. Perhaps some want to behave in between. Torch has some particularly inventive edging policies for pooling that have valid invalid regions and invalid invalid regions beyond the edges of the image.

Whatever the behavior, you will use the MPSNNFilter.paddingPolicy property to configure behavior. In its simplest form, a paddingPolicy is a object (possibly written by you, though MPS provides some) that conforms to the MPSNNPadding protocol. It should at minimum provide a padding method, which codes for common methods to size the result image, how to center it on the input image and where to place the remainder in cases where the image size isn't exactly divisible by the stride. This is a bitfield. You can use:

```
[MPSNNDefaultPadding paddingWithMethod: MPSNNPaddingMethodAlign... |
      MPSNNPaddingMethodAddRemainderTo...
                                        MPSNNPaddingMethodSize... ];
```

To quickly configure one of these. The filters also have a default padding policy, which may be appropriate most of the time.

Occasionally, something fancy needs to be done. In that case, the padding policy should set the MPSNNPaddingMethodCustom bit and implement the optional destinationImageDescriptorForSourceImages: sourceStates:forKernel:suggestedDescriptor: method. The MPSNNGraph will use the MPSNNPadding.paddingMethod to generate an initial guess for the configuration of the MPSCNNKernel.offset and the size and formatting of the result image and hand that to you in the form of a MPSImageDescriptor. You can modify the descriptor or the kernel (also passed to you) in your custom destinationImageDescriptorForSourceImages:sourceStates: forKernel:suggestedDescriptor: method, or just ignore it and make a new descriptor.

1.12 MPSNNGraph intermediate image sizing and centering

Typically the graph will make MPSTemporaryImages for these, based on the MPSImageDescriptor obtained from the padding policy. Temporary images alias one another and can be used to save a lot of memory, in the same way that malloc saves memory in your application by allowing you to reserve memory for a time, use it, then free it for reuse for something else. Ideally, most of the storage in your graph should be temporary images.

Because temporary images don't (shouldn't) last long, and can't be read by the CPU, some images probably can't be temporary. By default, the final image returned from the graph is not temporary. (See MPSNNGraph.destinationImageAllocator to adjust). Also, you may request that certain intermediate images be non-temporary so that you can access their contents from outside the graph using the MPSNNImageNode.exportFromGraph property.

Temporary images often take up almost no additional memory. Regular images always do. Some large graphs will only be able to run using temporary memory, as regular images would overwhelm the machine. Even if you allocate all your images up front and reuse them over and over, you will still very likely use much more memory with regular images, than if you just allocate temporary images as needed. Because temporary images do not generally allocate large amounts of storage, they are much cheaper and faster to use.

What kind of image is created after each filter node can be adjusted using the MPSNNImageNode.imageAllocator property. Two standard allocators are provided as defaultAllocator (MPSImage) and defaultAllocator (MPSTemporaryImage). You may of course write your own. This might be necessary for example if you wish to maintain your own MTLHeap and allocate from it.

1.13 MPSNNGraph debugging tips

In typical usage, some refinement, especially of padding policies, may be required to get the expected answer from MPS. If the result image is the wrong size, padding is typically the problem. When the answers are incorrect, the MPSCNNKernel.offset or other property may be incorrectly configured at some stage. As the graph is generated starting from an output image node, you may create other graphs starting at any image node within the graph. This will give you a view into the result produced from each intermediate layer with a minimum of fuss. In addition, the usual NSObject -debugDescription method is available to inspect objects to make sure they conform to expectation.

Note that certain operations such as neuron filters that follow convolution filters and image concatenation may be optimized away by the MPSNNGraph when it is constructed. The convolution can do neuron operations as part of its operation. Concatenation is best done by writing the result of earlier filter passes in the right place using MPSCNNKernel.destinationFeatureChannelOffset rather than by adding an extra copy. Other optimizations may be added as framework capabilities improve.

1.14 Sample Code

```
#import <MetalPerformanceShaders/MetalPerformanceShaders.h>

// Blur the input texture (in place if possible) on MTLCommandQueue q, and return the new texture.
// This is a trivial example. It is not necessary or necessarily advised to enqueue a MPSKernel on
// its own MTLCommandBuffer  or using its own MTLComputeCommandEncoder. Group work together.
//
// Here we assume that you have already gotten a MTLDevice using MTLCreateSystemDefaultDevice() or
// MTLCopyAllDevices(), used it to create a MTLCommandQueue with MTLDevice.newCommandQueue, and
// similarly made textures with the device as needed.
void  MyBlurTextureInPlace( id <MTLTexture> __strong *inTexture, float blurRadius, id <MTLCommandQueue> q)
{
    // Create "the usual Metal objects".
    // MPS does not need a dedicated MTLCommandBuffer or MTLComputeCommandEncoder.
    // This is a trivial example. You should reuse the MTL objects you already have, if you have them.
    id <MTLDevice> device = q.device;
    id <MTLCommandBuffer> buffer = [q commandBuffer];

    // Create a MPS filter.
    MPSImageGaussianBlur *blur = [[MPSImageGaussianBlur alloc]
      initWithDevice: device];
    if( nil == blur )
        MyHandleError(kOutOfMemory);

    // Set all MPSKernel properties to taste.
    blur.sigma = blurRadius;
    // defaults are okay here for other MPSKernel properties. (clipRect, origin, edgeMode)

    // Attempt to do the work in place.  Since we provided a copyAllocator as an out-of-place
    // fallback, we don't need to check to see if it succeeded or not.
    [ blur encodeToCommandBuffer: commandBuffer
                  inPlaceTexture: inTexture         // may replace *inTexture
                   copyAllocator: myAllocator ];    // See MPSCopyAllocator definition for a sample
       myAllocator
    [ blur release];

    // the usual metal enqueue process
    [buffer waitUntilCompleted];     // slow!  Try enqueing more work on this or the next
                                     // command buffer instead of waiting every time.

    return result;
}
```

1.15 MPS Tuning Hints

MPS has been tuned for excellent performance across a diversity of devices and filter parameters. The tuning process focuses on minimizing both CPU and GPU latency for back to back calls on the same MTLCommmand↩ Buffer. It is possible, however, to inadvertently undo this optimization effort by introducing costly operations into the pipeline around the MPS filter, leading to disappointing overall results.

Here are some elements of good practice to avoid common pitfalls:

1. Don't wait for results to complete before enqueuing more work. There can be a significant delay (up to 2.5 ms) just to get an empty MTLCommandBuffer through the pipeline to where [MTLCommandBuffer waitUntil↩ Completed] returns. Instead, start encoding the next command buffer(s) while you wait for the first one to complete. Enqueue them too, so they can start immediately after the previous one exits the GPU. Don't wait for the CPU kernel to notice the first command buffer is done and start taking it apart and eventually make a callback to userland before beginning work on encoding the next one. By allowing the CPU and GPU to work concurrently in this way, throughput can be enhanced by up to a factor of ten.
2. There is a large cost to allocating buffers and textures. The cost can swamp the CPU, preventing you from keeping the GPU busy. Try to preallocate and reuse MTLResource objects as much as possible. The MPS↩ TemporaryImage may be used instead for short-lived dynamic allocations.
3. There is a cost to switching between render and compute encoders. Each time a new render encoder is used, there can be a substantial GPU mode switch cost that may undermine your throughput. To avoid the cost, try to batch compute work together. Since making a new MTLCommandBuffer forces you to make a new MTLCommandEncoder too, try to do more work with fewer MTLCommandBuffers.

4. On currently available iOS devices we find that for some image operations, particularly those involving multiple passes - for example, if you are chaining multiple MPS image filters together — performance can be improved by up a factor of two by breaking the work into tiles about 512 kB in size. Use -sourceRegionForDestination↩ Size: to find the MPSRegion needed for each tile.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

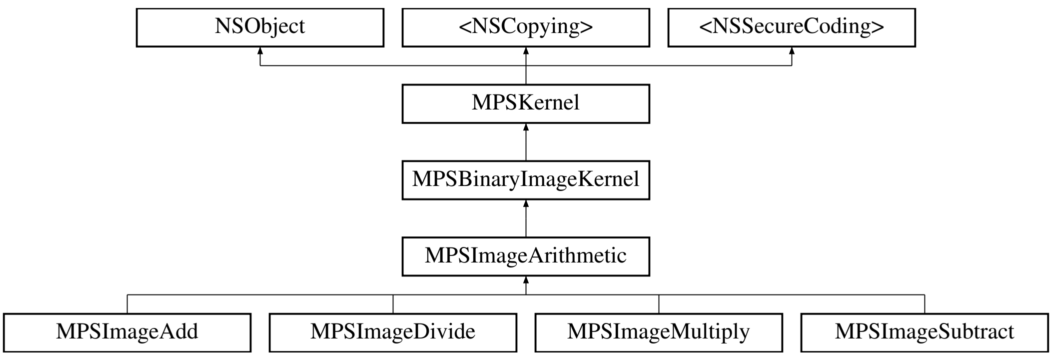
Chapter 5

Class Documentation

5.1 MPSBinaryImageKernel Class Reference

```
#import <MPSImageKernel.h>
```

Inheritance diagram for MPSBinaryImageKernel:



Instance Methods

- (nonnull instancetype) - initWithDevice:
- (nullable instancetype) - initWithCoder:device:
- (BOOL) - encodeToCommandBuffer:primaryTexture:inPlaceSecondaryTexture:fallbackCopyAllocator:
- (BOOL) - encodeToCommandBuffer:inPlacePrimaryTexture:secondaryTexture:fallbackCopyAllocator:
- (void) - encodeToCommandBuffer:primaryTexture:secondaryTexture:destinationTexture:
- (void) - encodeToCommandBuffer:primaryImage:secondaryImage:destinationImage:
- (MPSRegion) - primarySourceRegionForDestinationSize:
- (MPSRegion) - secondarySourceRegionForDestinationSize:

Properties

- MPSOffset primaryOffset
- MPSOffset secondaryOffset
- MPSImageEdgeMode primaryEdgeMode
- MPSImageEdgeMode secondaryEdgeMode
- MTLRegion clipRect

Additional Inherited Members

5.1.1 Detailed Description

This depends on Metal.framework A MPSBinaryImageKernel consumes two MTLTextures and produces one MTLTexture.

5.1.2 Method Documentation

5.1.2.1 encodeToCommandBuffer:inPlacePrimaryTexture:secondaryTexture:fallbackCopyAllocator:()

```
- (BOOL) encodeToCommandBuffer:
            (nonnull id< MTLCommandBuffer >) commandBuffer
            inPlacePrimaryTexture:(__nonnull id< MTLTexture > __strong *__nonnull) inPlacePrimaryTexture
            secondaryTexture:(nonnull id< MTLTexture >) secondaryTexture
            fallbackCopyAllocator:(nullable MPSCopyAllocator) copyAllocator
```

Attempt to apply a MPSKernel to a texture in place. This method attempts to apply the MPSKernel in place on a texture.

```
In-place operation means that the same texture is used both to hold the input
image and the results. Operating in-place can be an excellent way to reduce
resource utilization, and save time and energy. While simple Metal kernels can
not operate in place because textures can not be readable and writable at the
same time, some MPSKernels can operate in place because they use
multi-pass algorithms. Whether a MPSKernel can operate in-place can
depend on current hardware, operating system revision and the parameters
and properties passed to it. You should never assume that a MPSKernel will
continue to work in place, even if you have observed it doing so before.
```

If the operation succeeds in-place, YES is returned. If the in-place operation fails and no copyAllocator is provided, then NO is returned. In neither case is the pointer held at ∗texture modified.

Failure during in-place operation is common. You may find it simplifies your code to provide a copyAllocator. When an in-place filter fails, your copyAllocator will be invoked to create a new texture in which to write the results, allowing the filter to proceed reliably out-of-place. The original texture will be released, replaced with a pointer to the new texture and YES will be returned. If the allocator returns an invalid texture, it is released, ∗texture remains unmodified and NO is returned. Please see the MPSCopyAllocator definition for a sample allocator implementation.

Note: Image filters that look at neighboring pixel values may actually consume more memory when operating in place than out of place. Many such operations are tiled internally to save intermediate texture storage, but can not tile when operating in place. The memory savings for tiling is however very short term, typically the lifetime of the MTLCommandBuffer.

Parameters

commandBuffer	A valid MTLCommandBuffer to receive the encoded filter
inPlacePrimaryTexture	A pointer to a valid MTLTexture containing secondary image. On success, the image contents and possibly texture itself will be replaced with the result image.
secondaryTexture	A pointer to a valid MTLTexture containing the primary source image. It will not be overwritten.
copyAllocator	An optional block to allocate a new texture to hold the results, in case in-place
	operation is not possible. The allocator may use a different MTLPixelFormat or size than the original texture. You may enqueue operations on the provided MTLCommandBuffer using the provided MTLComputeCommandEncoder to initialize the texture contents.

Returns

On success, YES is returned. The texture may have been replaced with a new texture if a copyAllocator was provided. On failure, NO is returned. The texture is unmodified.

5.1.2.2 encodeToCommandBuffer:primaryImage:secondaryImage:destinationImage:()

```
- (void) encodeToCommandBuffer:
            (nonnull id< MTLCommandBuffer >) commandBuffer
            primaryImage:(MPSImage *__nonnull) primaryImage
            secondaryImage:(MPSImage *__nonnull) secondaryImage
            destinationImage:(MPSImage *__nonnull) destinationImage
```

Encode a MPSKernel into a command Buffer. The operation shall proceed out-of-place.

Parameters

commandBuffer	A valid MTLCommandBuffer to receive the encoded filter
primaryImage	A valid MPSImage containing the primary source image.
secondaryImage	A valid MPSImage containing the secondary source image.
destinationImage	A valid MPSImage to be overwritten by result image. destinationImage may not alias the source images.

5.1.2.3 encodeToCommandBuffer:primaryTexture:inPlaceSecondaryTexture:fallbackCopyAllocator:()

```
- (BOOL) encodeToCommandBuffer:
            (nonnull id< MTLCommandBuffer >) commandBuffer
            primaryTexture:(nonnull id< MTLTexture >) primaryTexture
            inPlaceSecondaryTexture:(__nonnull id< MTLTexture > __strong *__nonnull) in↩
PlaceSecondaryTexture
            fallbackCopyAllocator:(nullable MPSCopyAllocator) copyAllocator
```

This method attempts to apply the MPSKernel in place on a texture.

```
In-place operation means that the same texture is used both to hold the input
image and the results. Operating in-place can be an excellent way to reduce
resource utilization, and save time and energy. While simple Metal kernels can
not operate in place because textures can not be readable and writable at the
same time, some MPSKernels can operate in place because they use
multi-pass algorithms. Whether a MPSKernel can operate in-place can
depend on current hardware, operating system revision and the parameters
and properties passed to it. You should never assume that a MPSKernel will
continue to work in place, even if you have observed it doing so before.
```

If the operation succeeds in-place, YES is returned. If the in-place operation fails and no copyAllocator is provided, then NO is returned. In neither case is the pointer held at ∗texture modified.

Failure during in-place operation is common. You may find it simplifies your code to provide a copyAllocator. When an in-place filter fails, your copyAllocator will be invoked to create a new texture in which to write the results, allowing the filter to proceed reliably out-of-place. The original texture will be released, replaced with a pointer to the new texture and YES will be returned. If the allocator returns an invalid texture, it is released, ∗texture remains unmodified and NO is returned. Please see the MPSCopyAllocator definition for a sample allocator implementation.

Note: Image filters that look at neighboring pixel values may actually consume more memory when operating in place than out of place. Many such operations are tiled internally to save intermediate texture storage, but can not tile when operating in place. The memory savings for tiling is however very short term, typically the lifetime of the MTLCommandBuffer.

Attempt to apply a MPSKernel to a texture in place.

Parameters

commandBuffer	A valid MTLCommandBuffer to receive the encoded filter
primaryTexture	A pointer to a valid MTLTexture containing the primary source image. It will not be overwritten.
inPlaceSecondaryTexture	A pointer to a valid MTLTexture containing secondary image. On success, the image contents and possibly texture itself will be replaced with the result image.
copyAllocator	An optional block to allocate a new texture to hold the results, in case in-place operation is not possible. The allocator may use a different MTLPixelFormat or size than the original texture. You may enqueue operations on the provided MTLCommandBuffer using the provided MTLComputeCommandEncoder to initialize the texture contents.

Returns

On success, YES is returned. The texture may have been replaced with a new texture if a copyAllocator was provided. On failure, NO is returned. The texture is unmodified.

5.1.2.4 encodeToCommandBuffer:primaryTexture:secondaryTexture:destinationTexture:()

```
- (void) encodeToCommandBuffer:
            (nonnull id< MTLCommandBuffer >) commandBuffer
            primaryTexture:(nonnull id< MTLTexture >) primaryTexture
            secondaryTexture:(nonnull id< MTLTexture >) secondaryTexture
            destinationTexture:(nonnull id< MTLTexture >) destinationTexture
```

Encode a MPSKernel into a command Buffer. The operation shall proceed out-of-place.

Parameters

commandBuffer	A valid MTLCommandBuffer to receive the encoded filter
primaryTexture	A valid MTLTexture containing the primary source image.
secondaryTexture	A valid MTLTexture containing the secondary source image.
destinationTexture	A valid MTLTexture to be overwritten by result image. destinationTexture may not alias the source textures.

5.1.2.5 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSKernel.

5.1.2.6 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

a pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSKernel.

Reimplemented in MPSImageArithmetic, MPSImageAdd, MPSImageSubtract, MPSImageMultiply, and MPS←ImageDivide.

5.1.2.7 primarySourceRegionForDestinationSize:()

```
- (MPSRegion) primarySourceRegionForDestinationSize:
            (MTLSize) destinationSize
```

primarySourceRegionForDestinationSize: is used to determine which region of the primaryTexture will be read by encodeToCommandBuffer:primaryTexture:secondaryTexture:destinationTexture (and in-place variants) when the filter runs. This information may be needed if the primary source image is broken into multiple textures. The size of the full (untiled) destination image is provided. The region of the full (untiled) source image that will be read is returned. You can then piece together an appropriate texture containing that information for use in your tiled context.

The function will consult the MPSBinaryImageKernel primaryOffset and clipRect parameters, to determine the full region read by the function. Other parameters such as kernelHeight and kernelWidth will be consulted as necessary. All properties should be set to intended values prior to calling primarySourceRegionForDestinationSize:.

```
Caution: This function operates using global image coordinates, but
-encodeToCommandBuffer:... uses coordinates local to the source and
destination image textures. Consequently, the primaryOffset and clipRect
attached to this object will need to be updated using a global to
local coordinate transform before -encodeToCommandBuffer:... is
called.
```

Determine the region of the source texture that will be read for a encode operation

Parameters

destinationSize	The size of the full virtual destination image.

Returns

The area in the virtual source image that will be read.

5.1.2.8 secondarySourceRegionForDestinationSize:()

```
- (MPSRegion) secondarySourceRegionForDestinationSize:
            (MTLSize) destinationSize
```

secondarySourceRegionForDestinationSize: is used to determine which region of the sourceTexture will be read by encodeToCommandBuffer:primaryTexture:secondaryTexture:destinationTexture (and in-place variants) when the filter runs. This information may be needed if the secondary source image is broken into multiple textures. The size of the full (untiled) destination image is provided. The region of the full (untiled) secondary source image that will be read is returned. You can then piece together an appropriate texture containing that information for use in your tiled context.

The function will consult the MPSBinaryImageKernel secondaryOffset and clipRect parameters, to determine the full region read by the function. Other parameters such as kernelHeight and kernelWidth will be consulted as necessary. All properties should be set to intended values prior to calling secondarySourceRegionForDestinationSize:.

```
Caution: This function operates using global image coordinates, but
-encodeToCommandBuffer:... uses coordinates local to the source and
destination image textures. Consequently, the secondaryOffset and clipRect
attached to this object will need to be updated using a global to
local coordinate transform before -encodeToCommandBuffer:... is
called.
```

Determine the region of the source texture that will be read for a encode operation

Parameters

destinationSize	The size of the full virtual destination image.

Returns

The area in the virtual source image that will be read.

5.1.3 Property Documentation

5.1.3.1 clipRect

```
- clipRect  [read], [write], [nonatomic], [assign]
```

An optional clip rectangle to use when writing data. Only the pixels in the rectangle will be overwritten. A MTL↩
Region that indicates which part of the destination to overwrite. If the clipRect does not lie completely within the destination image, the intersection between clip rectangle and destination bounds is used. Default: MPSRectNoClip (MPSKernel::MPSRectNoClip) indicating the entire image.

See Also: MetalPerformanceShaders.h subsubsection_clipRect

5.1.3.2 primaryEdgeMode

```
- primaryEdgeMode  [read], [write], [nonatomic], [assign]
```

The MPSImageEdgeMode to use when texture reads stray off the edge of the primary source image Most MPS↩
Kernel objects can read off the edge of a source image. This can happen because of a negative offset property, because the offset + clipRect.size is larger than the source image or because the filter looks at neighboring pixels, such as a Convolution or morphology filter. Default: usually MPSImageEdgeModeZero. (Some MPSKernel types default to MPSImageEdgeModeClamp, because MPSImageEdgeModeZero is either not supported or would produce unexpected results.)

See Also: MetalPerformanceShaders.h subsubsection_edgemode

5.1.3.3 primaryOffset

```
- primaryOffset  [read], [write], [nonatomic], [assign]
```

The position of the destination clip rectangle origin relative to the primary source buffer. The offset is defined to be the position of clipRect.origin in source coordinates. Default: {0,0,0}, indicating that the top left corners of the clipRect and primary source image align.

See Also: MetalPerformanceShaders.h subsubsection_mpsoffset

5.1.3.4 secondaryEdgeMode

```
- secondaryEdgeMode [read], [write], [nonatomic], [assign]
```

The MPSImageEdgeMode to use when texture reads stray off the edge of the secondary source image Most MPSKernel objects can read off the edge of a source image. This can happen because of a negative offset property, because the offset + clipRect.size is larger than the source image or because the filter looks at neighboring pixels, such as a Convolution or morphology filter. Default: usually MPSImageEdgeModeZero. (Some MPSKernel types default to MPSImageEdgeModeClamp, because MPSImageEdgeModeZero is either not supported or would produce unexpected results.)

See Also: MetalPerformanceShaders.h subsubsection_edgemode

5.1.3.5 secondaryOffset

```
- secondaryOffset [read], [write], [nonatomic], [assign]
```

The position of the destination clip rectangle origin relative to the secondary source buffer. The offset is defined to be the position of clipRect.origin in source coordinates. Default: {0,0,0}, indicating that the top left corners of the clipRect and secondary source image align.

See Also: MetalPerformanceShaders.h subsubsection_mpsoffset

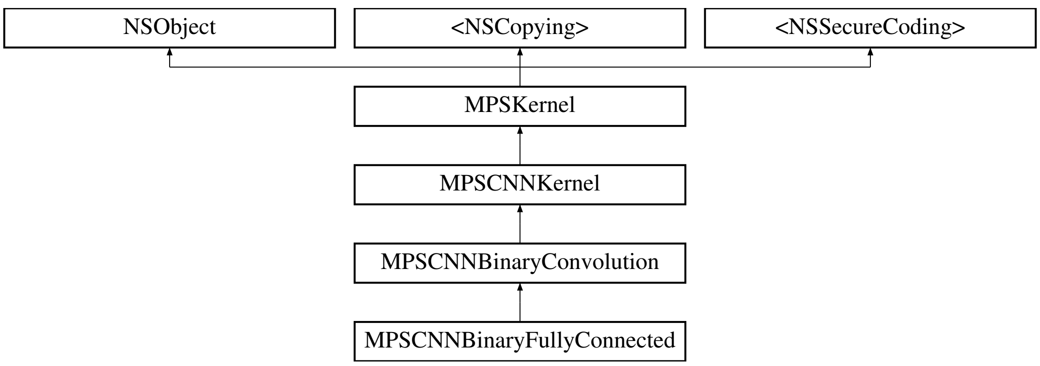
The documentation for this class was generated from the following file:

- MPSImageKernel.h

5.2 MPSCNNBinaryConvolution Class Reference

```
#import <MPSCNNConvolution.h>
```

Inheritance diagram for MPSCNNBinaryConvolution:



Instance Methods

- (nonnull instancetype) - initWithDevice:convolutionData:scaleValue:type:flags:
- (nonnull instancetype) - initWithDevice:convolutionData:outputBiasTerms:outputScaleTerms:inputBiasTerms:inputScaleTerms:type:flags:
- (nullable instancetype) - initWithCoder:device:
- (nonnull instancetype) - initWithDevice:

Properties

- NSUInteger inputFeatureChannels
- NSUInteger outputFeatureChannels

Additional Inherited Members

5.2.1 Detailed Description

This depends on Metal.framework The MPSCNNBinaryConvolution specifies a convolution with binary weights and an input image using binary approximations. The MPSCNNBinaryConvolution optionally first binarizes the input image and then convolves the result with a set of binary-valued filters, each producing one feature map in the output image (which is a normal image)

The output is computed as follows:

```
out[i, x, y, c] = ( sum_{dx,dy,f} in[i,x+dx, y+dy, f] x B[c,dx,dy,f] )
                    * scale[c] * beta[i,x,y] + bias[c], where
```

the sum over dx,dy is over the spatial filter kernel window defined by 'kernelWidth' and 'KernelHeight', sum over 'f' is over the input feature channel indices within group, 'B' contains the binary weights, interpreted as {-1,1} or { 0, 1 } and scale[c] is the 'outputScaleTerms' array and bias is the 'outputBiasTerms' array. Above 'i' is the image index in batch the sum over input channels 'f' runs through the group indices.

The convolution operator 'x' is defined by MPSCNNBinaryConvolutionType passed in at initialization time of the filter (

See also

initWithDevice). In case 'type' = MPSCNNBinaryConvolutionTypeBinaryWeights, the input image is not binarized at all and the convolution is computed interpreting the weights as [0, 1] -> { -1, 1 } with the given scaling terms. In case 'type' = MPSCNNBinaryConvolutionTypeXNOR the convolution is computed by first binarizing the input image using the sign function 'bin(x) = x $<$ 0 ? -1 : 1' and the convolution multiplication is done with the XNOR-operator !(x ^ y) = delta_xy = { (x==y) ? 1 : 0 }, and scaled according to the optional scaling operations. Note that we output the values of the bitwise convolutions to interval { -1, 1 }, which means that the output of the XNOR-operator is scaled implicitly as follows: r = 2 $*$ (!(x ^ y)) - 1 = { -1, 1 }. This means that for a dot-product of two 32-bit words the result is: r = 2 $*$ popcount(!(x ^ y)) - 32 = 32 - 2 $*$ popcount(x ^ y) = { -32, -30, ..., 30, 32 }. In case 'type' = MPSCNNBinaryConvolutionTypeAND the convolution is computed by first binarizing the input image using the sign function 'bin(x) = x $<$ 0 ? -1 : 1' and the convolution multiplication is done with the AND-operator (x & y) = delta_xy $*$ delta_x1 = { (x==y==1) ? 1 : 0 }. and scaled according to the optional scaling operations. Note that we output the values of the AND-operation is assumed to lie in { 0, 1 } interval and hence no more implicit scaling takes place. This means that for a dot-product of two 32-bit words the result is: r = popcount(x & y) = { 0, ..., 31, 32 }.

The input data can be pre-offset and scaled by providing the 'inputBiasTerms' and 'inputScaleTerms' parameters for the initialization functions and this can be used for example to accomplish batch normalization of the data. The scaling of input values happens before possible beta-image computation.

The parameter 'beta' above is an optional image which is used to compute scaling factors for each spatial position and image index. For the XNOR-Net based networks this is computed as follows: beta[i,x,y] = sum_{dx,dy} A[i, x+dx, y+dy] / (kx $*$ ky), where (dx,dy) are summed over the convolution filter window [-kx/2, (kx-1)/2], [-ky/2, (ky-1)/2] and A[i,x,y] = sum_{c} abs(in[i,x,y,c]) / Nc, where 'in' is the original input image (in full precision) and Nc is the number of input channels in the input image. Parameter 'beta' is not passed as input and to enable beta-scaling the user can provide 'MPSCNNBinaryConvolutionFlagsUseBetaScaling' in the flags parameter in the initialization functions.

Finally the normal activation neuron is applied and the result is written to the output image.

NOTE: MPSCNNBinaryConvolution does not currently support groups $>$ 1.

5.2.2 Method Documentation

5.2.2.1 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSCNNKernel.

Reimplemented in MPSCNNBinaryFullyConnected.

5.2.2.2 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

A pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSCNNKernel.

Reimplemented in MPSCNNBinaryFullyConnected.

5.2.2.3 initWithDevice:convolutionData:outputBiasTerms:outputScaleTerms:inputBiasTerms:inputScaleTerms:type:flags:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            convolutionData:(nonnull id< MPSCNNConvolutionDataSource >) convolutionData
            outputBiasTerms:(const float *__nullable) outputBiasTerms
            outputScaleTerms:(const float *__nullable) outputScaleTerms
            inputBiasTerms:(const float *__nullable) inputBiasTerms
            inputScaleTerms:(const float *__nullable) inputScaleTerms
            type:(MPSCNNBinaryConvolutionType) type
            flags:(MPSCNNBinaryConvolutionFlags) flags
```

Initializes a binary convolution kernel with binary weights as well as both pre and post scaling terms.

Parameters

device	The MTLDevice on which this MPSCNNBinaryConvolution filter will be used
convolutionData	A pointer to a object that conforms to the MPSCNNConvolutionDataSource protocol. The MPSCNNConvolutionDataSource protocol declares the methods that an instance of MPSCNNBinaryConvolution uses to obtain the weights and the convolution descriptor. Each entry in the convolutionData:weights array is a 32-bit unsigned integer value and each bit represents one filter weight (given in machine byte order). The featurechannel indices increase from the least significant bit within the 32-bits. The number of entries is = ceil(inputFeatureChannels/32.0) $*$ outputFeatureChannels $*$ kernelHeight $*$ kernelWidth The layout of filter weight is so that it can be reinterpreted as a 4D tensor (array) weight[outputChannels][kernelHeight][kernelWidth][ceil(inputChannels / 32.0)] (The ordering of the reduction from 4D tensor to 1D is per C convention. The index based on inputchannels varies most rapidly, followed by kernelWidth, then kernelHeight and finally outputChannels varies least rapidly.)
outputBiasTerms	A pointer to bias terms to be applied to the convolution output. Each entry is a float value. The number of entries is = numberOfOutputFeatureMaps. If nil then 0.0 is used for bias. The values stored in the pointer are copied in and the array can be freed after this function returns.
outputScaleTerms	A pointer to scale terms to be applied to binary convolution results per output feature channel. Each entry is a float value. The number of entries is = numberOfOutputFeatureMaps. If nil then 1.0 is used. The values stored in the pointer are copied in and the array can be freed after this function returns.
inputBiasTerms	A pointer to offset terms to be applied to the input before convolution and before input scaling. Each entry is a float value. The number of entries is 'inputFeatureChannels'. If NULL then 0.0 is used for bias. The values stored in the pointer are copied in and the array can be freed after this function returns.
inputScaleTerms	A pointer to scale terms to be applied to the input before convolution, but after input biasing. Each entry is a float value. The number of entries is 'inputFeatureChannels'. If nil then 1.0 is used. The values stored in the pointer are copied in and the array can be freed after this function returns.
type	What kind of binarization strategy is to be used.
flags	See documentation above and documentation of MPSCNNBinaryConvolutionFlags.

Returns

A valid MPSCNNBinaryConvolution object or nil, if failure.

Reimplemented in MPSCNNBinaryFullyConnected.

5.2.2.4 initWithDevice:convolutionData:scaleValue:type:flags:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            convolutionData:(nonnull id< MPSCNNConvolutionDataSource >) convolutionData
            scaleValue:(float) scaleValue
            type:(MPSCNNBinaryConvolutionType) type
            flags:(MPSCNNBinaryConvolutionFlags) flags
```

Initializes a binary convolution kernel with binary weights and a single scaling term.

Parameters

device	The MTLDevice on which this MPSCNNBinaryConvolution filter will be used
convolutionData	A pointer to a object that conforms to the MPSCNNConvolutionDataSource protocol. The MPSCNNConvolutionDataSource protocol declares the methods that an instance of MPSCNNBinaryConvolution uses to obtain the weights and bias terms as well as the convolution descriptor. Each entry in the convolutionData:weights array is a 32-bit unsigned integer value and each bit represents one filter weight (given in machine byte order). The featurechannel indices increase from the least significant bit within the 32-bits. The number of entries is = ceil(inputFeatureChannels/32.0) ∗ outputFeatureChannels ∗ kernelHeight ∗ kernelWidth The layout of filter weight is so that it can be reinterpreted as a 4D tensor (array) weight[outputChannels][kernelHeight][kernelWidth][ceil(inputChannels / 32.0)] (The ordering of the reduction from 4D tensor to 1D is per C convention. The index based on inputchannels varies most rapidly, followed by kernelWidth, then kernelHeight and finally outputChannels varies least rapidly.)
scaleValue	A floating point value used to scale the entire convolution.
type	What kind of binarization strategy is to be used.
flags	See documentation above and documentation of MPSCNNBinaryConvolutionFlags.

Returns

A valid MPSCNNBinaryConvolution object or nil, if failure.

Reimplemented in MPSCNNBinaryFullyConnected.

5.2.3 Property Documentation

5.2.3.1 inputFeatureChannels

```
- (NSUInteger) inputFeatureChannels [read], [nonatomic], [assign]
```

5.2.3.2 outputFeatureChannels

```
- outputFeatureChannels [read], [nonatomic], [assign]
```

The number of feature channels per pixel in the output image.

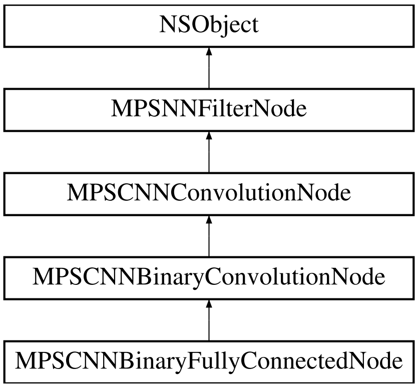
The documentation for this class was generated from the following file:

- MPSCNNConvolution.h

5.3 MPSCNNBinaryConvolutionNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSCNNBinaryConvolutionNode:



Instance Methods

- (nonnull instancetype) - initWithSource:weights:scaleValue:type:flags:

Class Methods

- (nonnull instancetype) + nodeWithSource:weights:scaleValue:type:flags:

Properties

- MPSCNNConvolutionStateNode ∗ convolutionState

5.3.1 Detailed Description

A MPSNNFilterNode representing a MPSCNNBinaryConvolution kernel

5.3.2 Method Documentation

5.3.2.1 initWithSource:weights:scaleValue:type:flags:()

```
- (nonnull instancetype) initWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            weights:(nonnull id< MPSCNNConvolutionDataSource >) weights
            scaleValue:(float) scaleValue
            type:(MPSCNNBinaryConvolutionType) type
            flags:(MPSCNNBinaryConvolutionFlags) flags
```

Init a node representing a MPSCNNBinaryConvolution kernel

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter
weights	A pointer to a valid object conforming to the MPSCNNConvolutionDataSource protocol. This object is provided by you to encapsulate storage for convolution weights and biases.
scaleValue	A floating point value used to scale the entire convolution.
type	What kind of binarization strategy is to be used.
flags	See documentation of MPSCNNBinaryConvolutionFlags.

Returns

A new MPSNNFilter node for a MPSCNNBinaryConvolution kernel.

Implemented in MPSCNNBinaryFullyConnectedNode.

5.3.2.2 nodeWithSource:weights:scaleValue:type:flags:()

```
+ (nonnull instancetype) nodeWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            weights:(nonnull id< MPSCNNConvolutionDataSource >) weights
            scaleValue:(float) scaleValue
            type:(MPSCNNBinaryConvolutionType) type
            flags:(MPSCNNBinaryConvolutionFlags) flags
```

Init an autoreleased not representing a MPSCNNBinaryConvolution kernel

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter
weights	A pointer to a valid object conforming to the MPSCNNConvolutionDataSource protocol. This object is provided by you to encapsulate storage for convolution weights and biases.
scaleValue	A floating point value used to scale the entire convolution.
type	What kind of binarization strategy is to be used.
flags	See documentation of MPSCNNBinaryConvolutionFlags.

Returns

A new MPSNNFilter node for a MPSCNNBinaryConvolution kernel.

Implemented in MPSCNNBinaryFullyConnectedNode.

5.3.3 Property Documentation

5.3.3.1 convolutionState

```
- (MPSCNNConvolutionStateNode*) convolutionState [read], [nonatomic], [assign]
```

unavailable

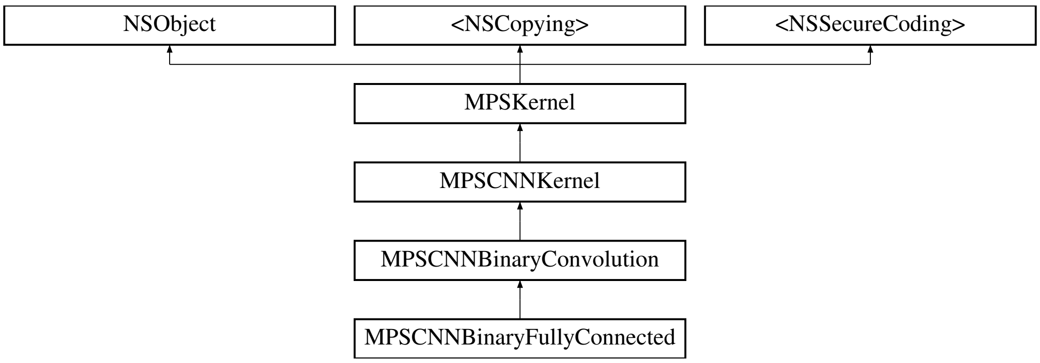
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.4 MPSCNNBinaryFullyConnected Class Reference

```
#import <MPSCNNConvolution.h>
```

Inheritance diagram for MPSCNNBinaryFullyConnected:



Instance Methods

- (nonnull instancetype) - initWithDevice:convolutionData:scaleValue:type:flags:
- (nonnull instancetype) - initWithDevice:convolutionData:outputBiasTerms:outputScaleTerms:inputBias↩Terms:inputScaleTerms:type:flags:
- (nullable instancetype) - initWithCoder:device:
- (nonnull instancetype) - initWithDevice:

Additional Inherited Members

5.4.1 Detailed Description

This depends on Metal.framework The MPSCNNBinaryFullyConnected specifies a fully connected convolution layer with binary weights and optionally binarized input image. See MPSCNNFullyConnected for details on the fully connected layer and MPSCNNBinaryConvolution for binary convolutions.

The default padding policy for MPSCNNBinaryConvolution is different from most filters. It uses MPSNNPadding↩MethodSizeValidOnly instead of MPSNNPaddingMethodSizeSame.

5.4.2 Method Documentation

5.4.2.1 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSCNNBinaryConvolution.

5.4.2.2 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

A pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSCNNBinaryConvolution.

5.4.2.3 initWithDevice:convolutionData:outputBiasTerms:outputScaleTerms:inputBiasTerms:inputScaleTerms:type:flags:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            convolutionData:(nonnull id< MPSCNNConvolutionDataSource >) convolutionData
            outputBiasTerms:(const float *__nullable) outputBiasTerms
            outputScaleTerms:(const float *__nullable) outputScaleTerms
            inputBiasTerms:(const float *__nullable) inputBiasTerms
            inputScaleTerms:(const float *__nullable) inputScaleTerms
            type:(MPSCNNBinaryConvolutionType) type
            flags:(MPSCNNBinaryConvolutionFlags) flags
```

Initializes a binary fully connected kernel with binary weights as well as both pre and post scaling terms.

Parameters

device	The MTLDevice on which this MPSCNNBinaryFullyConnected filter will be used
convolutionData	A pointer to a object that conforms to the MPSCNNConvolutionDataSource protocol. The MPSCNNConvolutionDataSource protocol declares the methods that an instance of MPSCNNBinaryFullyConnected uses to obtain the weights and the convolution descriptor. Each entry in the convolutionData:weights array is a 32-bit unsigned integer value and each bit represents one filter weight (given in machine byte order). The featurechannel indices increase from the least significant bit within the 32-bits. The number of entries is = ceil(inputFeatureChannels/32.0) ∗ outputFeatureChannels ∗ kernelHeight ∗ kernelWidth The layout of filter weight is so that it can be reinterpreted as a 4D tensor (array) weight[outputChannels][kernelHeight][kernelWidth][ceil(inputChannels / 32.0)] (The ordering of the reduction from 4D tensor to 1D is per C convention. The index based on inputchannels varies most rapidly, followed by kernelWidth, then kernelHeight and finally outputChannels varies least rapidly.)
outputBiasTerms	A pointer to bias terms to be applied to the convolution output. Each entry is a float value. The number of entries is = numberOfOutputFeatureMaps. If nil then 0.0 is used for bias. The values stored in the pointer are copied in and the array can be freed after this function returns.
outputScaleTerms	A pointer to scale terms to be applied to binary convolution results per output feature channel. Each entry is a float value. The number of entries is = numberOfOutputFeatureMaps. If nil then 1.0 is used. The values stored in the pointer are copied in and the array can be freed after this function returns.
inputBiasTerms	A pointer to offset terms to be applied to the input before convolution and before input scaling. Each entry is a float value. The number of entries is 'inputFeatureChannels'. If NULL then 0.0 is used for bias. The values stored in the pointer are copied in and the array can be freed after this function returns.
inputScaleTerms	A pointer to scale terms to be applied to the input before convolution, but after input biasing. Each entry is a float value. The number of entries is 'inputFeatureChannels'. If nil then 1.0 is used. The values stored in the pointer are copied in and the array can be freed after this function returns.
type	What kind of binarization strategy is to be used.
flags	See documentation above and documentation of MPSCNNBinaryConvolutionFlags.

Returns

A valid MPSCNNBinaryFullyConnected object or nil, if failure.

Reimplemented from MPSCNNBinaryConvolution.

5.4.2.4 initWithDevice:convolutionData:scaleValue:type:flags:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            convolutionData:(nonnull id< MPSCNNConvolutionDataSource >) convolutionData
            scaleValue:(float) scaleValue
            type:(MPSCNNBinaryConvolutionType) type
            flags:(MPSCNNBinaryConvolutionFlags) flags
```

Initializes a binary fully connected kernel with binary weights and a single scaling term.

Parameters

device	The MTLDevice on which this MPSCNNBinaryFullyConnected filter will be used
convolutionData	A pointer to a object that conforms to the MPSCNNConvolutionDataSource protocol. The MPSCNNConvolutionDataSource protocol declares the methods that an instance of MPSCNNBinaryFullyConnected uses to obtain the weights and bias terms as well as the convolution descriptor. Each entry in the convolutionData:weights array is a 32-bit unsigned integer value and each bit represents one filter weight (given in machine byte order). The featurechannel indices increase from the least significant bit within the 32-bits. The number of entries is = ceil(inputFeatureChannels/32.0) $*$ outputFeatureChannels $*$ kernelHeight $*$ kernelWidth The layout of filter weight is so that it can be reinterpreted as a 4D tensor (array) weight[outputChannels][kernelHeight][kernelWidth][ceil(inputChannels / 32.0)] (The ordering of the reduction from 4D tensor to 1D is per C convention. The index based on inputchannels varies most rapidly, followed by kernelWidth, then kernelHeight and finally outputChannels varies least rapidly.)
scaleValue	A single floating point value used to scale the entire convolution. Each entry is a float value. The number of entries is 'inputFeatureChannels'. If nil then 1.0 is used.
type	What kind of binarization strategy is to be used.
flags	See documentation above and documentation of MPSCNNBinaryConvolutionFlags.

Returns

A valid MPSCNNBinaryFullyConnected object or nil, if failure.

Reimplemented from MPSCNNBinaryConvolution.

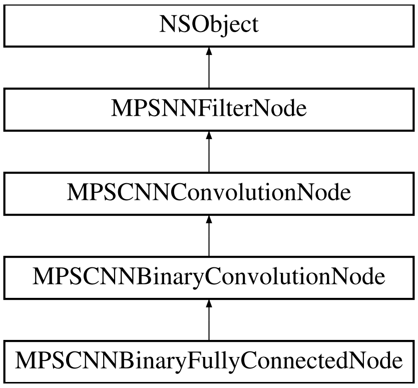
The documentation for this class was generated from the following file:

- MPSCNNConvolution.h

5.5 MPSCNNBinaryFullyConnectedNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSCNNBinaryFullyConnectedNode:



Instance Methods

- (nonnull instancetype) - initWithSource:weights:scaleValue:type:flags:

Class Methods

- (nonnull instancetype) + nodeWithSource:weights:scaleValue:type:flags:

Additional Inherited Members

5.5.1 Detailed Description

A MPSNNFilterNode representing a MPSCNNBinaryFullyConnected kernel

5.5.2 Method Documentation

5.5.2.1 initWithSource:weights:scaleValue:type:flags:()

```
- (nonnull instancetype) initWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            weights:(nonnull id< MPSCNNConvolutionDataSource >) weights
            scaleValue:(float) scaleValue
            type:(MPSCNNBinaryConvolutionType) type
            flags:(MPSCNNBinaryConvolutionFlags) flags
```

Init a node representing a MPSCNNBinaryFullyConnected kernel

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter
weights	A pointer to a valid object conforming to the MPSCNNConvolutionDataSource protocol. This object is provided by you to encapsulate storage for convolution weights and biases.
scaleValue	A floating point value used to scale the entire convolution.
type	What kind of binarization strategy is to be used.
flags	See documentation of MPSCNNBinaryConvolutionFlags.

Returns

A new MPSNNFilter node for a MPSCNNBinaryFullyConnected kernel.

Implements MPSCNNBinaryConvolutionNode.

5.5.2.2 nodeWithSource:weights:scaleValue:type:flags:()

```
+ (nonnull instancetype) nodeWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            weights:(nonnull id< MPSCNNConvolutionDataSource >) weights
            scaleValue:(float) scaleValue
            type:(MPSCNNBinaryConvolutionType) type
            flags:(MPSCNNBinaryConvolutionFlags) flags
```

Init an autoreleased not representing a MPSCNNBinaryFullyConnected kernel

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter
weights	A pointer to a valid object conforming to the MPSCNNConvolutionDataSource protocol. This object is provided by you to encapsulate storage for convolution weights and biases.
scaleValue	A floating point value used to scale the entire convolution.
type	What kind of binarization strategy is to be used.
flags	See documentation of MPSCNNBinaryConvolutionFlags.

Returns

A new MPSNNFilter node for a MPSCNNBinaryFullyConnected kernel.

Implements MPSCNNBinaryConvolutionNode.

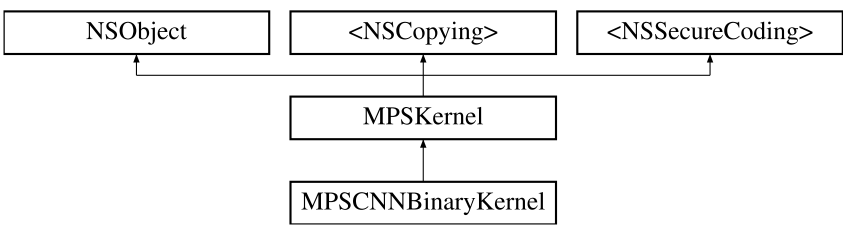
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.6 MPSCNNBinaryKernel Class Reference

```
#import <MPSCNNKernel.h>
```

Inheritance diagram for MPSCNNBinaryKernel:



Instance Methods

- (nonnull instancetype) - initWithDevice:
- (nullable instancetype) - initWithCoder:device:
- (void) - encodeToCommandBuffer:primaryImage:secondaryImage:destinationImage:
- (MPSImage ∗__nonnull) - encodeToCommandBuffer:primaryImage:secondaryImage:

Properties

- MPSOffset primaryOffset
- MPSOffset secondaryOffset
- MTLRegion clipRect
- NSUInteger destinationFeatureChannelOffset
- MPSImageEdgeMode primaryEdgeMode
- MPSImageEdgeMode secondaryEdgeMode
- NSUInteger kernelWidth
- NSUInteger kernelHeight
- NSUInteger primaryStrideInPixelsX
- NSUInteger primaryStrideInPixelsY
- NSUInteger secondaryStrideInPixelsX
- NSUInteger secondaryStrideInPixelsY
- BOOL isBackwards
- id< MPSNNPadding > padding
- id< MPSImageAllocator > destinationImageAllocator

Additional Inherited Members

5.6.1 Detailed Description

This depends on Metal.framework Describes a convolution neural network kernel. A MPSCNNKernel consumes two MPSImages, primary and secondary, and produces one MPSImage.

5.6.2 Method Documentation

5.6.2.1 encodeToCommandBuffer:primaryImage:secondaryImage:()

```
- (MPSImage * __nonnull) encodeToCommandBuffer:
            (nonnull id< MTLCommandBuffer >) commandBuffer
            primaryImage:(MPSImage *__nonnull) primaryImage
            secondaryImage:(MPSImage *__nonnull) secondaryImage
```

Encode a MPSCNNKernel into a command Buffer. Create a texture to hold the result and return it. In the first iteration on this method, encodeToCommandBuffer:sourceImage:destinationImage: some work was left for the developer to do in the form of correctly setting the offset property and sizing the result buffer. With the introduction of the padding policy (see padding property) the filter can do this work itself. If you would like to have some input into what sort of MPSImage (e.g. temporary vs. regular) or what size it is or where it is allocated, you may set the destinationImageAllocator to allocate the image yourself.

This method uses the MPSNNPadding padding property to figure out how to size the result image and to set the offset property. See discussion in MPSNeuralNetworkTypes.h.

Parameters

commandBuffer	The command buffer
primaryImage	A MPSImages to use as the primary source images for the filter.
secondaryImage	A MPSImages to use as the secondary source images for the filter.

Returns

A MPSImage or MPSTemporaryImage allocated per the destinationImageAllocator containing the output of the graph. The returned image will be automatically released when the command buffer completes. If you want to keep it around for longer, retain the image. (ARC will do this for you if you use it later.)

5.6.2.2 encodeToCommandBuffer:primaryImage:secondaryImage:destinationImage:()

```
- (void) encodeToCommandBuffer:
            (nonnull id< MTLCommandBuffer >) commandBuffer
            primaryImage:(MPSImage *__nonnull) primaryImage
            secondaryImage:(MPSImage *__nonnull) secondaryImage
            destinationImage:(MPSImage *__nonnull) destinationImage
```

Encode a MPSCNNKernel into a command Buffer. The operation shall proceed out-of-place. This is the older style of encode which reads the offset, doesn't change it, and ignores the padding method.

Parameters

commandBuffer	A valid MTLCommandBuffer to receive the encoded filter
primaryImage	A valid MPSImage object containing the primary source image.
secondaryImage	A valid MPSImage object containing the secondary source image.
destinationImage	A valid MPSImage to be overwritten by result image. destinationImage may not alias primarySourceImage or secondarySourceImage.

5.6.2.3 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSKernel.

5.6.2.4 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

A pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSKernel.

5.6.3 Property Documentation

5.6.3.1 clipRect

```
- clipRect  [read], [write], [nonatomic], [assign]
```

An optional clip rectangle to use when writing data. Only the pixels in the rectangle will be overwritten. A MTLRegion that indicates which part of the destination to overwrite. If the clipRect does not lie completely within the destination image, the intersection between clip rectangle and destination bounds is used. Default: MPSRectNoClip (MPSKernel::MPSRectNoClip) indicating the entire image. clipRect.origin.z is the index of starting destination image in batch processing mode. clipRect.size.depth is the number of images to process in batch processing mode.

See Also: MPSKernel clipRect

5.6.3.2 destinationFeatureChannelOffset

`- destinationFeatureChannelOffset [read], [write], [nonatomic], [assign]`

The number of channels in the destination MPSImage to skip before writing output. This is the starting offset into the destination image in the feature channel dimension at which destination data is written. This allows an application to pass a subset of all the channels in MPSImage as output of MPSKernel. E.g. Suppose MPSImage has 24 channels and a MPSKernel outputs 8 channels. If we want channels 8 to 15 of this MPSImage to be used as output, we can set destinationFeatureChannelOffset = 8. Note that this offset applies independently to each image when the MPSImage is a container for multiple images and the MPSCNNKernel is processing multiple images (clipRect.size.depth > 1). The default value is 0 and any value specifed shall be a multiple of 4. If MPSKernel outputs N channels, destination image MUST have at least destinationFeatureChannelOffset + N channels. Using a destination image with insufficient number of feature channels result in an error. E.g. if the MPSCNNConvolution outputs 32 channels, and destination has 64 channels, then it is an error to set destinationFeatureChannelOffset > 32.

5.6.3.3 destinationImageAllocator

`- (id<MPSImageAllocator>) destinationImageAllocator [read], [write], [nonatomic], [retain]`

Method to allocate the result image for -encodeToCommandBuffer:sourceImage: Default: defaultAllocator (MPS←TemporaryImage)

5.6.3.4 isBackwards

`- isBackwards [read], [nonatomic], [assign]`

YES if the filter operates backwards. This influences how strideInPixelsX/Y should be interpreted.

5.6.3.5 kernelHeight

`- kernelHeight [read], [nonatomic], [assign]`

The height of the MPSCNNKernel filter window This is the vertical diameter of the region read by the filter for each result pixel. If the MPSCNNKernel does not have a filter window, then 1 will be returned.

5.6.3.6 kernelWidth

`- kernelWidth [read], [nonatomic], [assign]`

The width of the MPSCNNKernel filter window This is the horizontal diameter of the region read by the filter for each result pixel. If the MPSCNNKernel does not have a filter window, then 1 will be returned.

5.6.3.7 padding

`- padding [read], [write], [nonatomic], [retain]`

The padding method used by the filter This influences how strideInPixelsX/Y should be interpreted. Default: MP←SNNPaddingMethodAlignCentered | MPSNNPaddingMethodAddRemainderToTopLeft | MPSNNPaddingMethod←SizeSame Some object types (e.g. MPSCNNFullyConnected) may override this default with something appropriate to its operation.

5.6.3.8 primaryEdgeMode

```
- primaryEdgeMode  [read], [write], [nonatomic], [assign]
```

The MPSImageEdgeMode to use when texture reads stray off the edge of the primary source image Most MPS↩ Kernel objects can read off the edge of the source image. This can happen because of a negative offset property, because the offset + clipRect.size is larger than the source image or because the filter looks at neighboring pixels, such as a Convolution filter. Default: MPSImageEdgeModeZero.

See Also: MPSKernelEdgeMode

5.6.3.9 primaryOffset

```
- primaryOffset  [read], [write], [nonatomic], [assign]
```

The position of the destination clip rectangle origin relative to the primary source buffer. The offset is defined to be the position of clipRect.origin in source coordinates. Default: {0,0,0}, indicating that the top left corners of the clipRect and primary source image align. offset.z is the index of starting source image in batch processing mode.

See Also: subsubsection_mpsoffset

5.6.3.10 primaryStrideInPixelsX

```
- primaryStrideInPixelsX  [read], [nonatomic], [assign]
```

The downsampling (or upsampling if a backwards filter) factor in the horizontal dimension for the primary source image If the filter does not do up or downsampling, 1 is returned.

5.6.3.11 primaryStrideInPixelsY

```
- primaryStrideInPixelsY  [read], [nonatomic], [assign]
```

The downsampling (or upsampling if a backwards filter) factor in the vertical dimension for the primary source image If the filter does not do up or downsampling, 1 is returned.

5.6.3.12 secondaryEdgeMode

```
- secondaryEdgeMode  [read], [write], [nonatomic], [assign]
```

The MPSImageEdgeMode to use when texture reads stray off the edge of the primary source image Most MPS↩ Kernel objects can read off the edge of the source image. This can happen because of a negative offset property, because the offset + clipRect.size is larger than the source image or because the filter looks at neighboring pixels, such as a Convolution filter. Default: MPSImageEdgeModeZero.

See Also: MPSKernelEdgeMode

5.6.3.13 secondaryOffset

```
- secondaryOffset [read], [write], [nonatomic], [assign]
```

The position of the destination clip rectangle origin relative to the secondary source buffer. The offset is defined to be the position of clipRect.origin in source coordinates. Default: {0,0,0}, indicating that the top left corners of the clipRect and secondary source image align. offset.z is the index of starting source image in batch processing mode.

See Also: subsubsection_mpsoffset

5.6.3.14 secondaryStrideInPixelsX

```
- secondaryStrideInPixelsX [read], [nonatomic], [assign]
```

The downsampling (or upsampling if a backwards filter) factor in the horizontal dimension for the secondary source image If the filter does not do up or downsampling, 1 is returned.

5.6.3.15 secondaryStrideInPixelsY

```
- secondaryStrideInPixelsY [read], [nonatomic], [assign]
```

The downsampling (or upsampling if a backwards filter) factor in the vertical dimension for the secondary source image If the filter does not do up or downsampling, 1 is returned.

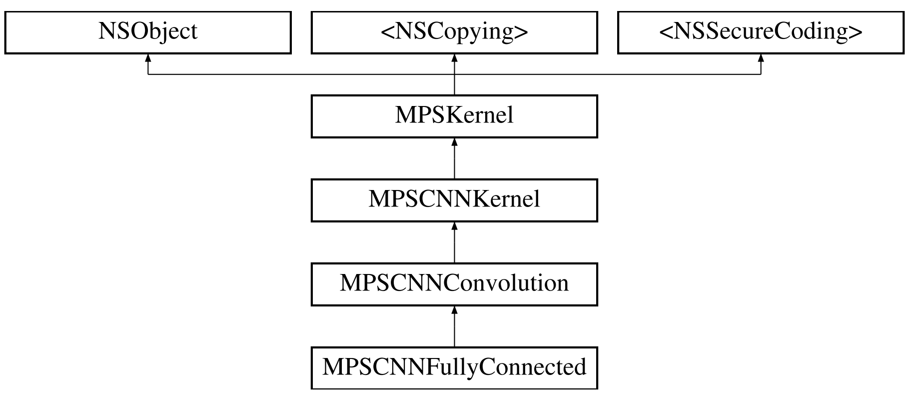
The documentation for this class was generated from the following file:

- MPSCNNKernel.h

5.7 MPSCNNConvolution Class Reference

```
#import <MPSCNNConvolution.h>
```

Inheritance diagram for MPSCNNConvolution:



Instance Methods

- (nonnull instancetype) - initWithDevice:convolutionDescriptor:kernelWeights:biasTerms:flags:
- (nonnull instancetype) - initWithDevice:weights:
- (nullable instancetype) - initWithCoder:device:
- (nonnull instancetype) - initWithDevice:
- (void) - encodeToCommandBuffer:sourceImage:destinationImage:state:

Properties

- NSUInteger inputFeatureChannels
- NSUInteger outputFeatureChannels
- NSUInteger groups
- NSUInteger subPixelScaleFactor
- NSUInteger dilationRate
- NSUInteger dilationRateX
- NSUInteger dilationRateY
- const MPSCNNNeuron *__nullable neuron
- const MPSCNNNeuron *__nullable MPSCNNNeuronType neuronType
- float neuronParameterA
- float neuronParameterB

Additional Inherited Members

5.7.1 Detailed Description

This depends on Metal.framework The MPSCNNConvolution specifies a convolution. The MPSCNNConvolution convolves the input image with a set of filters, each producing one feature map in the output image.

5.7.2 Method Documentation

5.7.2.1 encodeToCommandBuffer:sourceImage:destinationImage:state:()

```
- (void) encodeToCommandBuffer:
            (nonnull id< MTLCommandBuffer >) commandBuffer
            sourceImage:(MPSImage *__nonnull) sourceImage
            destinationImage:(MPSImage *__nonnull) destinationImage
            state:(MPSCNNConvolutionState *__nonnull *__nonnull) state
```

Encode a MPSCNNKernel into a command Buffer. The operation shall proceed out-of-place.

Parameters

commandBuffer	A valid MTLCommandBuffer to receive the encoded filter
sourceImage	A valid MPSImage object containing the source image.
destinationImage	A valid MPSImage to be overwritten by result image. destinationImage may not alias sourceImage.
state	state that is consumed by MPSCNNConvolutionTraspose to correctly size its destination image.

5.7.2.2 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSCNNKernel.

Reimplemented in MPSCNNFullyConnected.

5.7.2.3 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

A pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSCNNKernel.

Reimplemented in MPSCNNFullyConnected.

5.7.2.4 initWithDevice:convolutionDescriptor:kernelWeights:biasTerms:flags:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            convolutionDescriptor:(const MPSCNNConvolutionDescriptor *__nonnull) convolution←
Descriptor
            kernelWeights:(const float *__nonnull) kernelWeights
            biasTerms:(const float *__nullable) biasTerms
            flags:(MPSCNNConvolutionFlags) flags
```

Initializes a convolution kernel

Parameters

device	The MTLDevice on which this MPSCNNConvolution filter will be used
convolutionDescriptor	A pointer to a MPSCNNConvolutionDescriptor.
kernelWeights	A pointer to a weights array. Each entry is a float value. The number of entries is = inputFeatureChannels ∗ outputFeatureChannels ∗ kernelHeight ∗ kernelWidth The layout of filter weight is so that it can be reinterpreted as 4D tensor (array) weight[outputChannels][kernelHeight][kernelWidth][inputChannels / groups] Weights are converted to half float (fp16) internally for best performance.
biasTerms	A pointer to bias terms to be applied to the convolution output. Each entry is a float value. The number of entries is = numberOfOutputFeatureMaps
flags	Currently unused. Pass MPSCNNConvolutionFlagsNone

Returns

A valid MPSCNNConvolution object or nil, if failure.

Reimplemented in MPSCNNFullyConnected.

5.7.2.5 initWithDevice:weights:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            weights:(nonnull id< MPSCNNConvolutionDataSource >) weights
```

Initializes a convolution kernel

Parameters

device	The MTLDevice on which this MPSCNNConvolution filter will be used
weights	A pointer to a object that conforms to the MPSCNNConvolutionDataSource protocol. The MPSCNNConvolutionDataSource protocol declares the methods that an instance of MPSCNNConvolution uses to obtain the weights and bias terms for the CNN convolution filter.

Returns

A valid MPSCNNConvolution object or nil, if failure.

Reimplemented in MPSCNNFullyConnected.

5.7.3 Property Documentation

5.7.3.1 dilationRate

```
- dilationRate  [read], [nonatomic], [assign]
```

Dilation rate which was passed in as part of MPSCNNConvolutionDescriptor when creating this MPSCNN↩ Convolution object. WARNING: This is deprecated. Please use dilationRateX and dilationRateY instead.

Dilation rate which was passed in as part of MPSCNNConvolutionDescriptor when creating this MPSCNN↩ Convolution object.

5.7.3.2 dilationRateX

```
- dilationRateX  [read], [nonatomic], [assign]
```

Dilation rate which was passed in as part of MPSCNNConvolutionDescriptor when creating this MPSCNN↩ Convolution object.

5.7.3.3 dilationRateY

```
- (NSUInteger) dilationRateY  [read], [nonatomic], [assign]
```

5.7.3.4 groups

```
- groups  [read], [nonatomic], [assign]
```

Number of groups input and output channels are divided into.

5.7.3.5 inputFeatureChannels

```
- inputFeatureChannels  [read], [nonatomic], [assign]
```

The number of feature channels per pixel in the input image.

5.7.3.6 neuron

```
- neuron  [read], [nonatomic], [assign]
```

MPSCNNNeuron filter to be applied as part of convolution. Can be nil in wich case no neuron activation fuction is applied.

5.7.3.7 neuronParameterA

```
- (float) neuronParameterA [read], [nonatomic], [assign]
```

Parameter "a" for the neuron. Default: 1.0f Please see class description for interpretation of a.

5.7.3.8 neuronParameterB

```
- (float) neuronParameterB [read], [nonatomic], [assign]
```

Parameter "b" for the neuron. Default: 1.0f Please see class description for interpretation of b.

5.7.3.9 neuronType

```
- (const MPSCNNNeuron* __nullable MPSCNNNeuronType) neuronType [read], [nonatomic], [assign]
```

The type of neuron to append to the convolution Please see class description for a full list. Default is MPSCNN↩ NeuronTypeNone.

5.7.3.10 outputFeatureChannels

```
- outputFeatureChannels [read], [nonatomic], [assign]
```

The number of feature channels per pixel in the output image.

5.7.3.11 subPixelScaleFactor

```
- subPixelScaleFactor [read], [nonatomic], [assign]
```

Sub pixel scale factor which was passed in as part of MPSCNNConvolutionDescriptor when creating this MPSC↩ NNConvolution object.

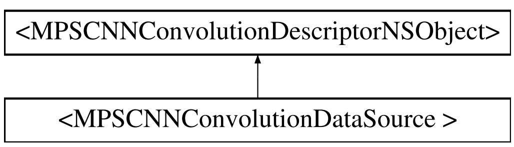
The documentation for this class was generated from the following file:

- MPSCNNConvolution.h

5.8 <MPSCNNConvolutionDataSource > Protocol Reference

```
#import <MPSCNNConvolution.h>
```

Inheritance diagram for <MPSCNNConvolutionDataSource >:



Instance Methods

- (MPSDataType) - dataType
- (MPSCNNConvolutionDescriptor ∗__nonnull) - descriptor
- (void ∗__nonnull) - weights
- (float ∗__nullable) - biasTerms
- (BOOL) - load
- (void) - purge
- (NSString ∗__nullable) - label
- (vector_float2 ∗__nonnull) - rangesForUInt8Kernel
- (float ∗__nonnull) - lookupTableForUInt8Kernel

5.8.1 Method Documentation

5.8.1.1 biasTerms()

```
- (float * __nullable MPSCNNConvolutionDataSource) biasTerms [required]
```

Returns a pointer to the bias terms for the convolution. Each entry in the array is a single precision IEEE-754 float and represents one bias. The number of entries is equal to outputFeatureChannels.

Frequently, this function is a single line of code to return a pointer to memory allocated in -load. It may also just return nil.

Note: bias terms are always float, even when the weights are not.

5.8.1.2 dataType()

```
- (MPSDataType MPSCNNConvolutionDataSource) dataType [required]
```

Alerts MPS what sort of weights are provided by the object For MPSCNNConvolution, MPSDataTypeUInt8, MP↩ SDataTypeFloat16 and MPSDataTypeFloat32 are supported for normal convolutions using MPSCNNConvolution. MPSCNNBinaryConvolution assumes weights to be of type MPSDataTypeUInt32 always.

5.8.1.3 descriptor()

```
- (MPSCNNConvolutionDescriptor * __nonnull MPSCNNConvolutionDataSource) descriptor [required]
```

Return a MPSCNNConvolutionDescriptor as needed MPS will not modify this object other than perhaps to retain it. User should set the appropriate neuron in the creation of convolution descriptor and for batch normalization use:

```
-setBatchNormalizationParametersForInferenceWithMean:variance:gamma:beta:epsilon:
```

Returns

A MPSCNNConvolutionDescriptor that describes the kernel housed by this object.

5.8.1.4 label()

```
- (NSString*__nullable MPSCNNConvolutionDataSource) label    [required]
```

A label that is transferred to the convolution at init time Overridden by a MPSCNNConvolutionNode.label if it is non-nil.

5.8.1.5 load()

```
- (BOOL MPSCNNConvolutionDataSource) load    [required]
```

Alerts the data source that the data will be needed soon Each load alert will be balanced by a purge later, when MPS no longer needs the data from this object. Load will always be called atleast once after initial construction or each purge of the object before anything else is called.

Returns

Returns YES on success. If NO is returned, expect MPS object construction to fail.

5.8.1.6 lookupTableForUInt8Kernel()

```
- (float * __nonnull MPSCNNConvolutionDataSource) lookupTableForUInt8Kernel    [optional]
```

A pointer to a 256 entry lookup table containing the values to use for the weight range [0,255]

5.8.1.7 purge()

```
- (void MPSCNNConvolutionDataSource) purge    [required]
```

Alerts the data source that the data is no longer needed Each load alert will be balanced by a purge later, when MPS no longer needs the data from this object.

5.8.1.8 rangesForUInt8Kernel()

```
- (vector_float2 * __nonnull MPSCNNConvolutionDataSource) rangesForUInt8Kernel    [optional]
```

A list of per-output channel limits that describe the 8-bit range This returns a pointer to an array of vector_↩ float2[outputChannelCount] values. The first value in the vector is the minimum value in the range. The second value in the vector is the maximum value in the range.

The 8-bit weight value is interpreted as:

```
float unorm8_weight = uint8_weight / 255.0f;    // unorm8_weight has range [0,1.0]
float max = range[outputChannel].y;
float min = range[outputChannel].x;
float weight = unorm8_weight * (max - min) + min
```

5.8.1.9 weights()

```
- (void * __nonnull MPSCNNConvolutionDataSource) weights [required]
```

Returns a pointer to the weights for the convolution. The type of each entry in array is given by -dataType. The number of entries is equal to:

```
inputFeatureChannels * outputFeatureChannels * kernelHeight * kernelWidth
```

The layout of filter weight is as a 4D tensor (array) weight[outputChannels][kernelHeight][kernelWidth][input←Channels / groups]

Frequently, this function is a single line of code to return a pointer to memory allocated in -load.

Batch normalization parameters are set using -descriptor.

Note: For binary-convolutions the layout of the weights are: weight[outputChannels][kernelHeight][kernelWidth][floor((inputChannels/groups)+31) / 32] with each 32 sub input feature channel index specified in machine byte order, so that for example the 13th feature channel bit can be extracted using bitmask = (1U << 13).

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

- MPSCNNConvolution.h

5.9 <MPSCNNConvolutionDataSource> Protocol Reference

```
#include <MPSCNNConvolution.h>
```

5.9.1 Detailed Description

Provides convolution filter weights and bias terms The MPSCNNConvolutionDataSource protocol declares the methods that an instance of MPSCNNConvolution uses to obtain the weights and bias terms for the CNN convolution filter.

Why? CNN weights can be large. If multiple copies of all the weights for all the convolutions are available unpacked in memory at the same time, some devices can run out of memory. The MPSCNNConvolutionDataSource is used to encapsulate a reference to the weights such as a file path, so that unpacking can be deferred until needed, then purged soon thereafter so that not all of the data must be in memory at the same time. MPS does not provide a class that conforms to this protocol. It is up to the developer to craft his own to encapsulate his data.

Batch normalization and the neuron activation function are handled using the -descriptor method.

Thread safety: The MPSCNNConvolutionDataSource object can be called by threads that are not the main thread. If you will be creating multiple MPSNNGraph objects concurrently in multiple threads and these share MPSCNN←ConvolutionDataSources, then the data source objects may be called reentrantly.

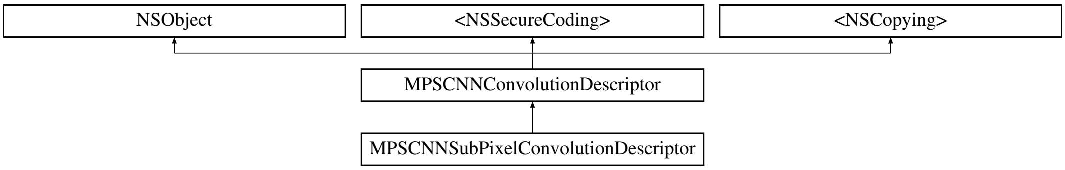
The documentation for this protocol was generated from the following file:

- MPSCNNConvolution.h

5.10 MPSCNNConvolutionDescriptor Class Reference

```
#import <MPSCNNConvolution.h>
```

Inheritance diagram for MPSCNNConvolutionDescriptor:



Instance Methods

- (void) - encodeWithCoder:
- (nullable instancetype) - initWithCoder:
- (void) - setBatchNormalizationParametersForInferenceWithMean:variance:gamma:beta:epsilon:

Class Methods

- (nonnull instancetype) + cnnConvolutionDescriptorWithKernelWidth:kernelHeight:inputFeatureChannels:outputFeatureChannels:neuronFilter:
- (nonnull instancetype) + cnnConvolutionDescriptorWithKernelWidth:kernelHeight:inputFeatureChannels:outputFeatureChannels:

Properties

- NSUInteger kernelWidth
- NSUInteger kernelHeight
- NSUInteger inputFeatureChannels
- NSUInteger outputFeatureChannels
- NSUInteger strideInPixelsX
- NSUInteger strideInPixelsY
- NSUInteger groups
- NSUInteger dilationRate
- NSUInteger dilationRateX
- NSUInteger dilationRateY
- const MPSCNNNeuron ∗__nullable neuron
- const MPSCNNNeuron ∗__nullable MPSCNNNeuronType neuronType
- float neuronParameterA
- float neuronParameterB
- BOOL supportsSecureCoding

5.10.1 Detailed Description

This depends on Metal.framework The MPSCNNConvolutionDescriptor specifies a convolution descriptor

5.10.2 Method Documentation

5.10.2.1 cnnConvolutionDescriptorWithKernelWidth:kernelHeight:inputFeatureChannels:outputFeatureChannels:()

```
+ (nonnull instancetype) cnnConvolutionDescriptorWithKernelWidth:
            (NSUInteger) kernelWidth
            kernelHeight:(NSUInteger) kernelHeight
            inputFeatureChannels:(NSUInteger) inputFeatureChannels
            outputFeatureChannels:(NSUInteger) outputFeatureChannels
```

Creates a convolution descriptor.

Parameters

kernelWidth	The width of the filter window. Must be > 0. Large values will take a long time.
kernelHeight	The height of the filter window. Must be > 0. Large values will take a long time.
inputFeatureChannels	The number of feature channels in the input image. Must be >= 1.
outputFeatureChannels	The number of feature channels in the output image. Must be >= 1.

Returns

A valid MPSCNNConvolutionDescriptor object or nil, if failure.

5.10.2.2 cnnConvolutionDescriptorWithKernelWidth:kernelHeight:inputFeatureChannels:outputFeatureChannels:neuronFilter:()

```
+ (nonnull instancetype) cnnConvolutionDescriptorWithKernelWidth:
            (NSUInteger) kernelWidth
            kernelHeight:(NSUInteger) kernelHeight
            inputFeatureChannels:(NSUInteger) inputFeatureChannels
            outputFeatureChannels:(NSUInteger) outputFeatureChannels
            neuronFilter:(const MPSCNNNeuron *__nullable) neuronFilter
```

This method is deprecated. Please use neuronType, neuronParameterA and neuronParameterB properites to fuse neuron with convolution.

Parameters

kernelWidth	The width of the filter window. Must be > 0. Large values will take a long time.
kernelHeight	The height of the filter window. Must be > 0. Large values will take a long time.
inputFeatureChannels	The number of feature channels in the input image. Must be >= 1.
outputFeatureChannels	The number of feature channels in the output image. Must be >= 1.
neuronFilter	An optional neuron filter that can be applied to the output of convolution.

Returns

A valid MPSCNNConvolutionDescriptor object or nil, if failure.

5.10.2.3 encodeWithCoder:()

```
- (void) encodeWithCoder:
            (NSCoder *__nonnull) aCoder
```

<NSSecureCoding> support

5.10.2.4 initWithCoder:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
```

<NSSecureCoding> support

5.10.2.5 setBatchNormalizationParametersForInferenceWithMean:variance:gamma:beta:epsilon:()

```
- (void) setBatchNormalizationParametersForInferenceWithMean:
            (const float *__nonnull) mean
            variance:(const float *__nonnull) variance
            gamma:(const float *__nullable) gamma
            beta:(const float *__nullable) beta
            epsilon:(const float) epsilon
```

Adds batch normalization for inference, it copies all the float arrays provided, expecting outputFeatureChannels elements in each.

This method will be used to pass in batch normalization parameters to the convolution during the init call. For inference we modify weights and bias going in convolution or Fully Connected layer to combine and optimize the layers.

```
w: weights for a corresponding output feature channel
b: bias for a corresponding output feature channel
W: batch normalized weights for a corresponding output feature channel
B: batch normalized bias for a corresponding output feature channel


I = gamma / sqrt(variance + epsilon), J = beta - ( I * mean )

W = w * I
B = b * I + J

Every convolution has (OutputFeatureChannel * kernelWidth * kernelHeight * InputFeatureChannel) weight

I, J are calculated, for every output feature channel separately to get the corresponding weights and
Thus, I, J are calculated and then used for every (kernelWidth * kernelHeight * InputFeatureChannel)
weights, and this is done OutputFeatureChannel number of times for each output channel.

thus, internally, batch normalized weights are computed as:

W[no][i][j][ni] = w[no][i][j][ni] * I[no]
```

```
no: index into outputFeatureChannel
i : index into kernel Height
j : index into kernel Width
ni: index into inputFeatureChannel

One usually doesn't see a bias term and batch normalization together as batch normalization potentiall
out the bias term after training, but in MPS if the user provides it, batch normalization will use the
formula to incorporate it, if user does not have bias terms then put a float array of zeroes in the co
init for bias terms of each output feature channel.


this comes from:
https://arxiv.org/pdf/1502.03167v3.pdf
```

Parameters

mean	Pointer to an array of floats of mean for each output feature channel
variance	Pointer to an array of floats of variance for each output feature channel
gamma	Pointer to an array of floats of gamma for each output feature channel
beta	Pointer to an array of floats of beta for each output feature channel
epsilon	A small float value used to have numerical stability in the code

5.10.3 Property Documentation

5.10.3.1 dilationRate

```
- dilationRate  [read], [write], [nonatomic], [assign]
```

dilationRate property can be used to implement dilated convolution as described in https://arxiv.org/pdf/1511.07122v3.pdf to aggregate global information in dense prediction problems. Dilated convolution is equivalent to performing convolution with dilated kernel of dimension kW_Dilated x kH_Dilated where

```
kW_Dilated = (kW-1)*dilationRate + 1
kH_Dilated = (kH-1)*dilationRate + 1
```

A dilated kernel is formed by taking the origianl kW x kH kernel and inserting d-1 zeros between consecutive entries in each row and d-1 zero filled rows between consecutive rows. The kernel is centered based on kW_Dilated and kH_Dilated.

When dilationRate = 1, it reduces to regular convolution. Default value is 1. WARNING: Property dilationRate is deprecated and simply sets dilationRateX = dilationRateY = dilationRate Users should use new dilationRateX and dilationRateY properties to set dilation rate in each dimension independently.

5.10.3.2 dilationRateX

```
- dilationRateX  [read], [write], [nonatomic], [assign]
```

dilationRateX property can be used to implement dilated convolution as described in https://arxiv.org/pdf/1511.07122v3.pdf to aggregate global information in dense prediction problems. Default value is 1. When set to value > 1, original kernel width, kW is dilated to

```
kW_Dilated = (kW-1)*dilationRateX + 1
```

by inserting d-1 zeros between consecutive entries in each row of the original kernel. The kernel is centered based on kW_Dilated.

5.10.3.3 dilationRateY

```
- dilationRateY [read], [write], [nonatomic], [assign]
```

dilationRateY property can be used to implement dilated convolution as described in https://arxiv.org/pdf/1511.07122v3.pdf to aggregate global information in dense prediction problems. Default value is 1. When set to value $>$ 1, original kernel height, kH is dilated to

```
kH_Dilated = (kH-1)*dilationRateY + 1
```

by inserting d-1 rows of zeros between consecutive row of the original kernel. The kernel is centered based on kH_Dilated.

5.10.3.4 groups

```
- groups [read], [write], [nonatomic], [assign]
```

Number of groups input and output channels are divided into. The default value is 1. Groups lets you reduce the parameterization. If groups is set to n, input is divided into n groups with inputFeatureChannels/n channels in each group. Similarly output is divided into n groups with outputFeatureChannels/n channels in each group. ith group in input is only connected to ith group in output so number of weights (parameters) needed is reduced by factor of n. Both inputFeatureChannels and outputFeatureChannels must be divisible by n and number of channels in each group must be multiple of 4.

5.10.3.5 inputFeatureChannels

```
- inputFeatureChannels [read], [write], [nonatomic], [assign]
```

The number of feature channels per pixel in the input image.

5.10.3.6 kernelHeight

```
- kernelHeight [read], [write], [nonatomic], [assign]
```

The height of the filter window. The default value is 3. Any positive non-zero value is valid, including even values. The position of the top edge of the filter window is given by offset.y - (kernelHeight$>>$1)

5.10.3.7 kernelWidth

```
- kernelWidth [read], [write], [nonatomic], [assign]
```

The width of the filter window. The default value is 3. Any positive non-zero value is valid, including even values. The position of the left edge of the filter window is given by offset.x - (kernelWidth$>>$1)

5.10.3.8 neuron

```
- neuron [read], [write], [nonatomic], [retain]
```

MPSCNNNeuron filter to be applied as part of convolution. This is applied after BatchNormalization in the end. Default is nil. This is deprecated. You dont need to create MPSCNNNeuron object to fuse with convolution. Use neuron properties in this descriptor.

5.10.3.9 neuronParameterA

```
- (float) neuronParameterA [read], [write], [nonatomic], [assign]
```

Parameter "a" for the neuron. Default: 1.0f Please see class description for interpretation of a.

5.10.3.10 neuronParameterB

```
- (float) neuronParameterB [read], [write], [nonatomic], [assign]
```

Parameter "b" for the neuron. Default: 1.0f Please see class description for interpretation of b.

5.10.3.11 neuronType

```
- (const MPSCNNNeuron* __nullable MPSCNNNeuronType) neuronType [read], [write], [nonatomic],
[assign]
```

The type of neuron to append to the convolution Please see class description for a full list. Default is MPSCNN←NeuronTypeNone.

5.10.3.12 outputFeatureChannels

```
- outputFeatureChannels [read], [write], [nonatomic], [assign]
```

The number of feature channels per pixel in the output image.

5.10.3.13 strideInPixelsX

```
- strideInPixelsX [read], [write], [nonatomic], [assign]
```

The output stride (downsampling factor) in the x dimension. The default value is 1.

5.10.3.14 strideInPixelsY

```
- strideInPixelsY [read], [write], [nonatomic], [assign]
```

The output stride (downsampling factor) in the y dimension. The default value is 1.

5.10.3.15 supportsSecureCoding

```
- (BOOL) supportsSecureCoding [read], [atomic], [assign]
```

<NSSecureCoding> support

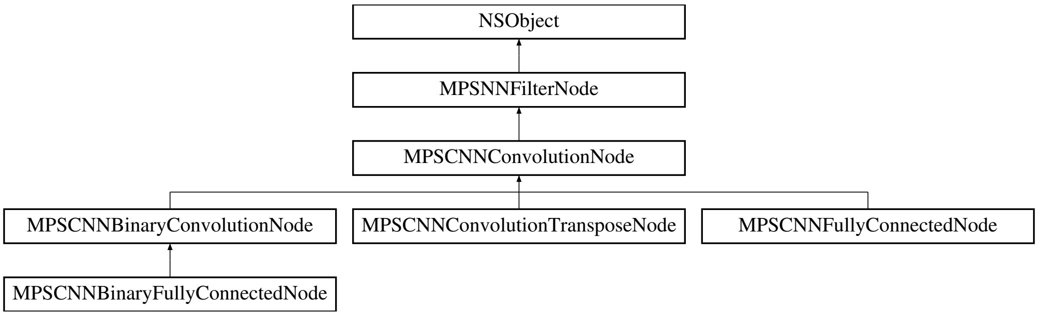
The documentation for this class was generated from the following file:

- MPSCNNConvolution.h

5.11 MPSCNNConvolutionNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSCNNConvolutionNode:



Instance Methods

- (nonnull instancetype) - initWithSource:weights:

Class Methods

- (nonnull instancetype) + nodeWithSource:weights:

Properties

- MPSCNNConvolutionStateNode ∗ convolutionState

5.11.1 Method Documentation

5.11.1.1 initWithSource:weights:()

```
- (nonnull instancetype) initWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            weights:(nonnull id< MPSCNNConvolutionDataSource >) weights
```

Init a node representing a MPSCNNConvolution kernel

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter
weights	A pointer to a valid object conforming to the MPSCNNConvolutionDataSource protocol. This object is provided by you to encapsulate storage for convolution weights and biases.

Returns

A new MPSNNFilter node for a MPSCNNConvolution kernel.

Implemented in MPSCNNFullyConnectedNode.

5.11.1.2 nodeWithSource:weights:()

```
+ (nonnull instancetype) nodeWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            weights:(nonnull id< MPSCNNConvolutionDataSource >) weights
```

Init an autoreleased not representing a MPSCNNConvolution kernel

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter
weights	A pointer to a valid object conforming to the MPSCNNConvolutionDataSource protocol. This object is provided by you to encapsulate storage for convolution weights and biases.

Returns

A new MPSNNFilter node for a MPSCNNConvolution kernel.

Implemented in MPSCNNFullyConnectedNode.

5.11.2 Property Documentation

5.11.2.1 convolutionState

```
- (MPSCNNConvolutionStateNode*) convolutionState [read], [nonatomic], [assign]
```

A node to represent a MPSCNNConvolutionState object Use this if the convolution is mirrored by a convolution transpose node later on in the graph to make sure that the size of the image returned from the convolution transpose matches the size of the image passed in to this node.

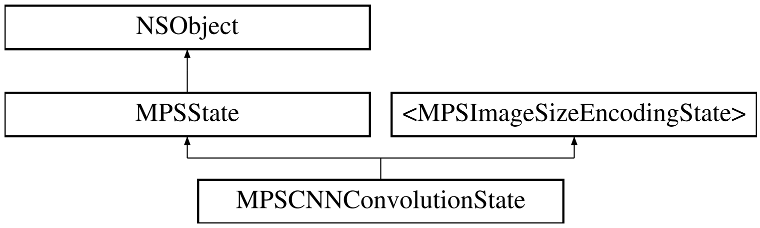
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.12 MPSCNNConvolutionState Class Reference

```
#import <MPSCNNConvolution.h>
```

Inheritance diagram for MPSCNNConvolutionState:



Additional Inherited Members

5.12.1 Detailed Description

The MPSCNNConvolutionState is returned by encode call of MPSCNNConvolution. It will be consumed by MP←SCNNConvolutionTranspose which needs size of source used by corresponding MPSCNNConvolution in forward pass to correctly size its destination. User is responsible for releasing it after it is consumed.

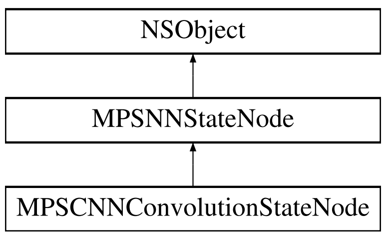
The documentation for this class was generated from the following file:

- MPSCNNConvolution.h

5.13 MPSCNNConvolutionStateNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSCNNConvolutionStateNode:



Additional Inherited Members

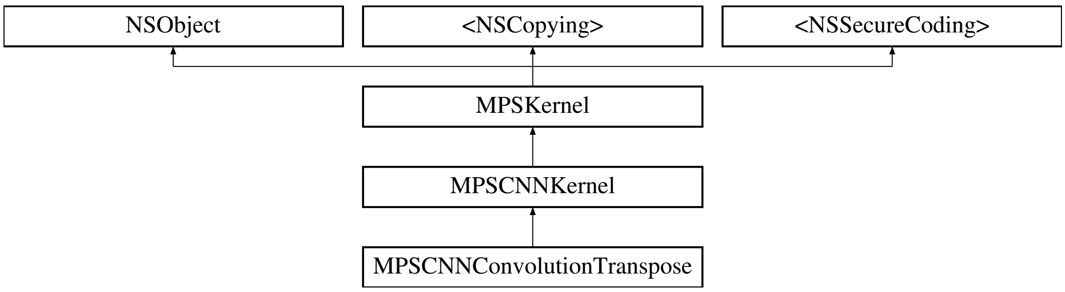
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.14 MPSCNNConvolutionTranspose Class Reference

```
#import <MPSCNNConvolution.h>
```

Inheritance diagram for MPSCNNConvolutionTranspose:



Instance Methods

- (nonnull instancetype) - initWithDevice:weights:
- (nonnull instancetype) - initWithDevice:
- (nullable instancetype) - initWithCoder:device:
- (void) - encodeToCommandBuffer:sourceImage:convolutionState:destinationImage:
- (MPSImage ∗__nonnull) - encodeToCommandBuffer:sourceImage:convolutionState:

Properties

- NSUInteger inputFeatureChannels
- NSUInteger outputFeatureChannels
- NSInteger kernelOffsetX
- NSInteger kernelOffsetY
- NSUInteger groups

Additional Inherited Members

5.14.1 Detailed Description

This depends on Metal.framework The MPSCNNConvolutionTranspose specifies a transposed convolution. The MPSCNNConvolutionTranspose convolves the input image with a set of filters, each producing one feature map in the output image.

When the stride in any dimension is greater than 1, the convolution transpose puts (stride - 1) zeroes in-between the source image pixels to create an expanded image. Then a convolution is done over the expanded image to generate the output of the convolution transpose.

Intermediate image size = (srcSize - 1) ∗ Stride + 1

Examples:

So in case of sride == 2 (this behaves same in both dimensions)

Source image:

| | | | | | 1 | 2 | 3 | 4 |

| | | | |

Intermediate Image:

| | | | | | | | | 1 | 0 | 2 | 0 | 3 | 0 | 4 |

| | | | | | | |

```
NOTE on Offset:
There are 2 types of offsets defined:
1) The Offset defined in MPSCNNKernel from which MPSCNNConvolutionTranspose inherits. This offset is a
   the kernel will be applied on the source.
2) The kernelOffsetX and kernelOffsetY which is the offset applied to the kernel when it is finally ap
   image.

So totalOffset = Offset * stride + kernelOffset

The offset defined by user refers to the coordinate frame of the expanded image
(we are showing only 1 dimension X it can be extended to Y dimension as well) :

X indicates where the convolution transpose begins:

Intermediate Image:  Offset = 0, kernelOffset = 0
 ___________________________
|   |   |   |   |   |   |   |
| 1 | 0 | 2 | 0 | 3 | 0 | 4 |
| X |   |   |   |   |   |   |
 ---------------------------


X indicates where the convolution transpose begins:

Intermediate Image:  Offset = 0, kernelOffset = 1
 ___________________________
|   |   |   |   |   |   |   |
| 1 | 0 | 2 | 0 | 3 | 0 | 4 |
|   | X |   |   |   |   |   |
 ---------------------------


X indicates where the convolution transpose begins:

Intermediate Image:  Offset = 0, kernelOffset = -1
   ___________________________
  |   |   |   |   |   |   |   |
X | 1 | 0 | 2 | 0 | 3 | 0 | 4 |
  |   |   |   |   |   |   |   |
   ---------------------------




So if the user wanted to apply an offset of 2 on the source image of convolution transpose:

Source image:
 _______________
|   |   |   |   |
| 1 | 2 | 3 | 4 |
|   |   | X |   |
 ---------------

offset = 2, kernelOffset = 0

Intermediate Image:
 ___________________________
|   |   |   |   |   |   |   |
| 1 | 0 | 2 | 0 | 3 | 0 | 4 |
|   |   |   |   | X |   |   |
 ---------------------------
```

5.14.2 Method Documentation

5.14.2.1 encodeToCommandBuffer:sourceImage:convolutionState:()

```
- (MPSImage * __nonnull) encodeToCommandBuffer:
            (nonnull id< MTLCommandBuffer >) commandBuffer
            sourceImage:(MPSImage *__nonnull) sourceImage
            convolutionState:(MPSCNNConvolutionState *__nullable) convolutionState
```

Encode a MPSCNNKernel into a command Buffer. Create a texture to hold the result and return it. In the first iteration on this method, encodeToCommandBuffer:sourceImage:destinationImage: some work was left for the developer to do in the form of correctly setting the offset property and sizing the result buffer. With the introduction of the padding policy (see padding property) the filter can do this work itself. If you would like to have some input into what sort of MPSImage (e.g. temporary vs. regular) or what size it is or where it is allocated, you may set the destinationImageAllocator to allocate the image yourself.

This method uses the MPSNNPadding padding property to figure out how to size the result image and to set the offset property. See discussion in MPSNeuralNetworkTypes.h.

Note: the regular encodeToCommandBuffer:sourceImage: method may be used when no state is needed, such as when the convolution transpose operation is not balanced by a matching convolution object upstream.

Parameters

commandBuffer	The command buffer
sourceImage	A MPSImage to use as the source images for the filter.
convolutionState	A valid MPSCNNConvolutionState from the MPSCNNConvoluton counterpart to this MPSCNNConvolutionTranspose. If there is no forward convolution counterpart, pass NULL here. This state affects the sizing the result.

Returns

A MPSImage or MPSTemporaryImage allocated per the destinationImageAllocator containing the output of the graph. The offset property will be adjusted to reflect the offset used during the encode. The returned image will be automatically released when the command buffer completes. If you want to keep it around for longer, retain the image. (ARC will do this for you if you use it later.)

5.14.2.2 encodeToCommandBuffer:sourceImage:convolutionState:destinationImage:()

```
- (void) encodeToCommandBuffer:
            (nonnull id< MTLCommandBuffer >) commandBuffer
            sourceImage:(MPSImage *__nonnull) sourceImage
            convolutionState:(MPSCNNConvolutionState *__nullable) convolutionState
            destinationImage:(MPSImage *__nonnull) destinationImage
```

Encode a MPSCNNKernel into a command Buffer. The operation shall proceed out-of-place. This is the older style of encode which reads the offset, doesn't change it, and ignores the padding method. Note: the regular encodeToCommandBuffer:sourceImage:destinationImage may be used when no state is needed, such as when the convolution transpose operation is not balanced by a matching convolution object upstream.

Parameters

commandBuffer	A valid MTLCommandBuffer to receive the encoded filter
sourceImage	A valid MPSImage object containing the source image.
convolutionState	A valid MPSCNNConvolutionState from the MPSCNNConvoluton counterpart to this MPSCNNConvolutionTranspose. If there is no forward convolution counterpart, pass NULL here.
destinationImage	A valid MPSImage to be overwritten by result image. destinationImage may not alias sourceImage.

5.14.2.3 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

<NSSecureCoding> support

Reimplemented from MPSCNNKernel.

5.14.2.4 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

A pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSCNNKernel.

5.14.2.5 initWithDevice:weights:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            weights:(nonnull id< MPSCNNConvolutionDataSource >) weights
```

Initializes a convolution kernel

Parameters

device	The MTLDevice on which this MPSCNNConvolutionTranspose filter will be used
weights	A pointer to a object that conforms to the MPSCNNConvolutionDataSource protocol. The MPSCNNConvolutionDataSource protocol declares the methods that an instance of MPSCNNConvolutionTranspose uses to obtain the weights and bias terms for the CNN convolutionTranspose filter. Currently we support only Float32 weights.

Returns

A valid MPSCNNConvolution object or nil, if failure.

5.14.3 Property Documentation

5.14.3.1 groups

```
- groups  [read], [nonatomic], [assign]
```

Number of groups input and output channels are divided into.

5.14.3.2 inputFeatureChannels

```
- inputFeatureChannels  [read], [nonatomic], [assign]
```

The number of feature channels per pixel in the input image.

5.14.3.3 kernelOffsetX

```
- kernelOffsetX  [read], [write], [nonatomic], [assign]
```

Offset in X from which the kernel starts sliding

5.14.3.4 kernelOffsetY

```
- kernelOffsetY  [read], [write], [nonatomic], [assign]
```

Offset in Y from which the kernel starts sliding

5.14.3.5 outputFeatureChannels

```
- outputFeatureChannels  [read], [nonatomic], [assign]
```

The number of feature channels per pixel in the output image.

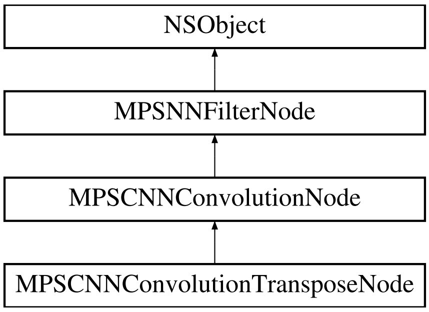
The documentation for this class was generated from the following file:

- MPSCNNConvolution.h

5.15 MPSCNNConvolutionTransposeNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSCNNConvolutionTransposeNode:



Instance Methods

- (nonnull instancetype) - initWithSource:convolutionState:weights:

Class Methods

- (nonnull instancetype) + nodeWithSource:convolutionState:weights:

Properties

- MPSCNNConvolutionStateNode ∗ convolutionState

5.15.1 Detailed Description

A MPSNNFilterNode representing a MPSCNNConvolutionTranspose kernel

5.15.2 Method Documentation

5.15.2.1 initWithSource:convolutionState:weights:()

```
- (nonnull instancetype) initWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            convolutionState:(MPSCNNConvolutionStateNode *__nullable) convolutionState
            weights:(nonnull id< MPSCNNConvolutionDataSource >) weights
```

Init a node representing a MPSCNNConvolutionTransposeNode kernel

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter
convolutionState	When the convolution transpose is used to 'undo' an earlier convolution in the graph, it is generally desired that the output image be the same size as the input image to the earlier convolution. You may optionally specify this size identity by passing in the MPSCNNConvolutionState node here.
weights	A pointer to a valid object conforming to the MPSCNNConvolutionDataSource protocol. This object is provided by you to encapsulate storage for convolution weights and biases.

Returns

A new MPSNNFilter node for a MPSCNNConvolutionTransposeNode kernel.

5.15.2.2 nodeWithSource:convolutionState:weights:()

```
+ (nonnull instancetype) nodeWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            convolutionState:(MPSCNNConvolutionStateNode *__nullable) convolutionState
            weights:(nonnull id< MPSCNNConvolutionDataSource >) weights
```

Init an autoreleased not representing a MPSCNNConvolutionTransposeNode kernel

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter
convolutionState	When the convolution transpose is used to 'undo' an earlier convolution in the graph, it is generally desired that the output image be the same size as the input image to the earlier convolution. You may optionally specify this size identity by passing in the MPSNNConvolutionStateNode created by the convolution node here.
weights	A pointer to a valid object conforming to the MPSCNNConvolutionDataSource protocol. This object is provided by you to encapsulate storage for convolution weights and biases.

Returns

A new MPSNNFilter node for a MPSCNNConvolutionTransposeNode kernel.

5.15.3 Property Documentation

5.15.3.1 convolutionState

```
- (MPSCNNConvolutionStateNode*) convolutionState [read], [nonatomic], [assign]
```

unavailable

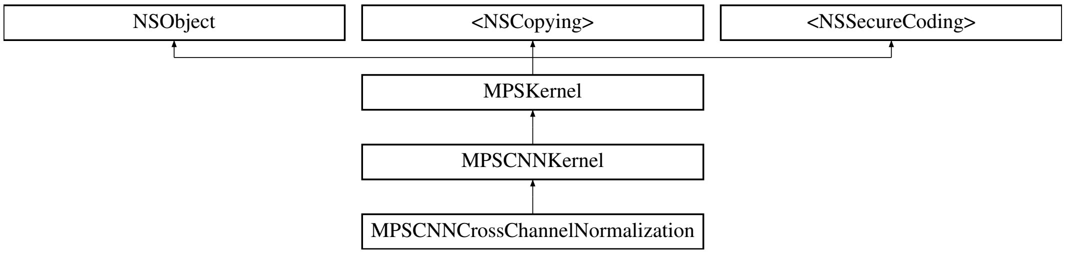
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.16 MPSCNNCrossChannelNormalization Class Reference

```
#import <MPSCNNNormalization.h>
```

Inheritance diagram for MPSCNNCrossChannelNormalization:



Instance Methods

- (nonnull instancetype) - initWithDevice:kernelSize:
- (nullable instancetype) - initWithCoder:device:
- (nonnull instancetype) - initWithDevice:

Properties

- float alpha
- float beta
- float delta
- NSUInteger kernelSize

Additional Inherited Members

5.16.1 Detailed Description

This depends on Metal.framework Specifies the normalization filter across feature channels. This normalization filter applies the filter to a local region across nearby feature channels, but with no spatial extent (i.e., they have shape kernelSize x 1 x 1). The normalized output is given by: Y(i,j,k) = X(i,j,k) / L(i,j,k)^beta, where the normalizing factor is: L(i,j,k) = delta + alpha/N * (sum_{q in Q(k)} X(i,j,q)^2, where N is the kernel size. The window Q(k) itself is defined as: Q(k) = [max(0, k-floor(N/2)), min(D-1, k+floor((N-1)/2)], where

k is the feature channel index (running from 0 to D-1) and D is the number of feature channels, and alpha, beta and delta are paremeters. It is the end-users responsibility to ensure that the combination of the parameters delta and alpha does not result in a situation where the denominator becomes zero - in such situations the resulting pixel-value is undefined.

5.16.2 Method Documentation

5.16.2.1 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSCNNKernel.

5.16.2.2 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

A pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSCNNKernel.

5.16.2.3 initWithDevice:kernelSize:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            kernelSize:(NSUInteger) kernelSize
```

Initialize a local response normalization filter in a channel

Parameters

device	The device the filter will run on
kernelSize	The kernel filter size in each dimension.

Returns

A valid MPSCNNCrossChannelNormalization object or nil, if failure.

5.16.3 Property Documentation

5.16.3.1 alpha

```
- alpha  [read], [write], [nonatomic], [assign]
```

The value of alpha. Default is 1.0. Must be non-negative.

5.16.3.2 beta

```
- beta  [read], [write], [nonatomic], [assign]
```

The value of beta. Default is 5.0

5.16.3.3 delta

```
- delta  [read], [write], [nonatomic], [assign]
```

The value of delta. Default is 1.0

5.16.3.4 kernelSize

```
- kernelSize  [read], [nonatomic], [assign]
```

The size of the square filter window. Default is 5

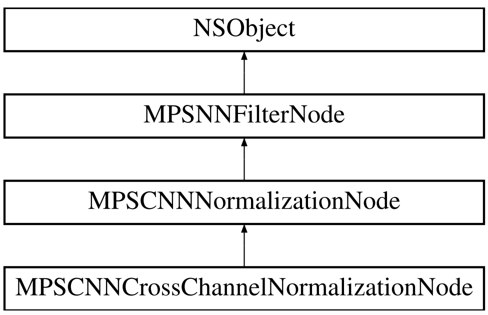
The documentation for this class was generated from the following file:

- MPSCNNNormalization.h

5.17 MPSCNNCrossChannelNormalizationNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSCNNCrossChannelNormalizationNode:



Instance Methods

- (nonnull instancetype) - initWithSource:kernelSize:
- (nonnull instancetype) - initWithSource:

Class Methods

- (nonnull instancetype) + nodeWithSource:kernelSize:

Properties

- NSUInteger kernelSizeInFeatureChannels

5.17.1 Method Documentation

5.17.1.1 initWithSource:()

```
- (nonnull instancetype) initWithSource:
             (MPSNNImageNode *__nonnull) sourceNode
```

Implements MPSCNNNormalizationNode.

5.17.1.2 initWithSource:kernelSize:()

```
- (nonnull instancetype) initWithSource:
             (MPSNNImageNode *__nonnull) sourceNode
             kernelSize:(NSUInteger) kernelSize
```

5.17.1.3 nodeWithSource:kernelSize:()

```
+ (nonnull instancetype) nodeWithSource:
             (MPSNNImageNode *__nonnull) sourceNode
             kernelSize:(NSUInteger) kernelSize
```

5.17.2 Property Documentation

5.17.2.1 kernelSizeInFeatureChannels

```
- (NSUInteger) kernelSizeInFeatureChannels [read], [write], [nonatomic], [assign]
```

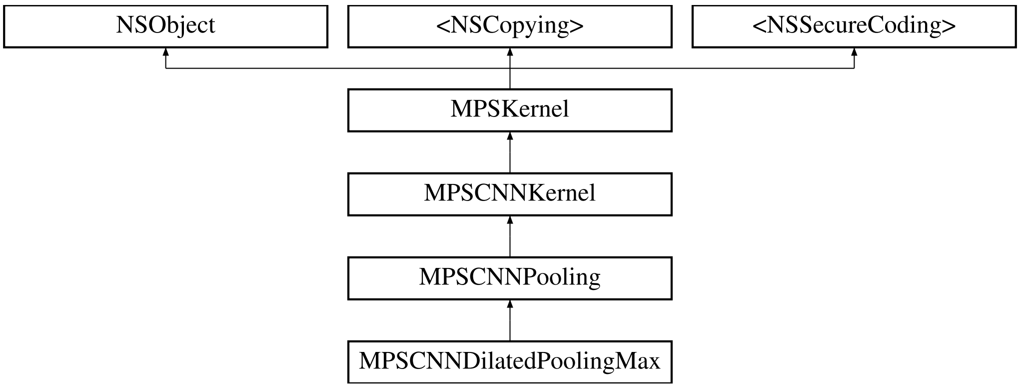
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.18 MPSCNNDilatedPoolingMax Class Reference

```
#import <MPSCNNPooling.h>
```

Inheritance diagram for MPSCNNDilatedPoolingMax:



Instance Methods

- (nonnull instancetype) - initWithDevice:kernelWidth:kernelHeight:dilationFactorX:dilationFactorY:strideInPixelsX:strideInPixelsY:
- (nonnull instancetype) - initWithDevice:kernelWidth:kernelHeight:dilationRateX:dilationRateY:strideInPixelsX:strideInPixelsY:
- (nullable instancetype) - initWithCoder:device:

Properties

- NSUInteger dilationFactorX
- NSUInteger dilationFactorY
- NSUInteger dilationRateX
- NSUInteger dilationRateY

Additional Inherited Members

5.18.1 Detailed Description

This depends on Metal.framework Specifies the dilated max pooling filter. For each pixel, returns the maximum value of pixels in the kernelWidth x kernelHeight filter region by step size dilationFactorX x dilationFactorY.

5.18.2 Method Documentation

5.18.2.1 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability See MPSKernel.h initWithCoder.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSCNNDilatedPoolingMax
device	The MTLDevice on which to make the MPSCNNDilatedPoolingMax

Returns

A new MPSCNNDilatedPoolingMax object, or nil if failure.

Reimplemented from MPSCNNPooling.

5.18.2.2 initWithDevice:kernelWidth:kernelHeight:dilationFactorX:dilationFactorY:strideInPixelsX:strideInPixelsY:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            kernelWidth:(NSUInteger) kernelWidth
            kernelHeight:(NSUInteger) kernelHeight
            dilationFactorX:(NSUInteger) dilationFactorX
            dilationFactorY:(NSUInteger) dilationFactorY
            strideInPixelsX:(NSUInteger) strideInPixelsX
            strideInPixelsY:(NSUInteger) strideInPixelsY
```

Initialize a MPSCNNDilatedPoolingMax pooling filter

Parameters

device	The device the filter will run on
kernelWidth	The width of the kernel. Can be an odd or even value.
kernelHeight	The height of the kernel. Can be an odd or even value.
dilationFactorX	The dilation factor in the x dimension.
dilationFactorY	The dilation factor in the y dimension.
strideInPixelsX	The output stride (downsampling factor) in the x dimension.
strideInPixelsY	The output stride (downsampling factor) in the y dimension.

Returns

A valid MPSCNNDilatedPoolingMax object or nil, if failure.

5.18.2.3 initWithDevice:kernelWidth:kernelHeight:dilationRateX:dilationRateY:strideInPixelsX:strideInPixelsY:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            kernelWidth:(NSUInteger) kernelWidth
            kernelHeight:(NSUInteger) kernelHeight
            dilationRateX:(NSUInteger) dilationRateX
            dilationRateY:(NSUInteger) dilationRateY
            strideInPixelsX:(NSUInteger) strideInPixelsX
            strideInPixelsY:(NSUInteger) strideInPixelsY
```

Initialize a MPSCNNDilatedPoolingMax pooling filter

Parameters

device	The device the filter will run on
kernelWidth	The width of the kernel. Can be an odd or even value.
kernelHeight	The height of the kernel. Can be an odd or even value.
dilationRateX	The dilation rate in the x dimension.
dilationRateY	The dilation rate in the y dimension.
strideInPixelsX	The output stride (downsampling factor) in the x dimension.
strideInPixelsY	The output stride (downsampling factor) in the y dimension.

Returns

A valid MPSCNNDilatedPoolingMax object or nil, if failure.

5.18.3 Property Documentation

5.18.3.1 dilationFactorX

```
- dilationFactorX [read], [nonatomic], [assign]
```

dilationFactorX for accessing the image passed in as source. NOTE: Deprecated, please use dilationRateX instead.

5.18.3.2 dilationFactorY

```
- dilationFactorY [read], [nonatomic], [assign]
```

dilationFactorY for accessing the image passed in as source. NOTE: Deprecated, please use dilationRateY instead.

5.18.3.3 dilationRateX

```
- dilationRateX  [read], [nonatomic], [assign]
```

dilationRateX for accessing the image passed in as source

5.18.3.4 dilationRateY

```
- dilationRateY  [read], [nonatomic], [assign]
```

dilationRateY for accessing the image passed in as source

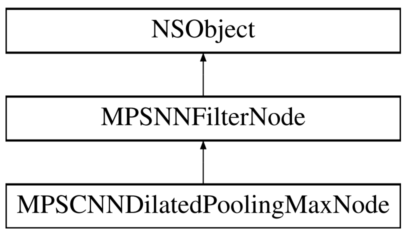
The documentation for this class was generated from the following file:

- MPSCNNPooling.h

5.19 MPSCNNDilatedPoolingMaxNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSCNNDilatedPoolingMaxNode:



Instance Methods

- (nonnull instancetype) - initWithSource:kernelWidth:kernelHeight:strideInPixelsX:strideInPixelsY:dilationFactorX:dilationFactorY:
- (nonnull instancetype) - initWithSource:kernelWidth:kernelHeight:strideInPixelsX:strideInPixelsY:dilationRateX:dilationRateY:
- (nonnull instancetype) - initWithSource:filterSize:stride:dilationFactor:
- (nonnull instancetype) - initWithSource:filterSize:stride:dilationRate:
- (nonnull instancetype) - initWithSource:filterSize:

Class Methods

- (nonnull instancetype) + nodeWithSource:filterSize:

Properties

- NSUInteger dilationFactorX
- NSUInteger NSUInteger dilationFactorY
- NSUInteger NSUInteger NSUInteger dilationRateX
- NSUInteger dilationRateY

5.19.1 Detailed Description

A node for a MPSCNNDilatedPooling kernel This class corresponds to the MPSCNNDilatedPooling class.

5.19.2 Method Documentation

5.19.2.1 initWithSource:filterSize:()

```
- (nonnull instancetype) initWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            filterSize:(NSUInteger) size
```

Convenience initializer for MPSCNNDilatedPooling nodes with square non-overlapping kernels

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter
size	kernelWidth = kernelHeight = strideInPixelsX = strideInPixelsY = dilationRateX = dilationRateY = size

Returns

A new MPSNNFilter node for a MPSCNNDilatedPooling kernel.

5.19.2.2 initWithSource:filterSize:stride:dilationFactor:()

```
- (nonnull instancetype) initWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            filterSize:(NSUInteger) size
            stride:(NSUInteger) stride
            dilationFactor:(NSUInteger) dilationFactor
```

Convenience initializer for MPSCNNDilatedPooling nodes with square kernels and equal dilation factors

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter
size	kernelWidth = kernelHeight = size
stride	strideInPixelsX = strideInPixelsY = stride
dilationFactor	dilationFactorX = dilationFactorY = stride

Returns

A new MPSNNFilter node for a MPSCNNDilatedPooling kernel.

5.19.2.3 initWithSource:filterSize:stride:dilationRate:()

```
- (nonnull instancetype) initWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            filterSize:(NSUInteger) size
            stride:(NSUInteger) stride
            dilationRate:(NSUInteger) dilationRate
```

Convenience initializer for MPSCNNDilatedPooling nodes with square kernels and equal dilation factors

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter
size	kernelWidth = kernelHeight = size
stride	strideInPixelsX = strideInPixelsY = stride
dilationRate	dilationRateX = dilationRateY = stride

Returns

A new MPSNNFilter node for a MPSCNNDilatedPooling kernel.

5.19.2.4 initWithSource:kernelWidth:kernelHeight:strideInPixelsX:strideInPixelsY:dilationFactorX:dilationFactorY:()

```
- (nonnull instancetype) initWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            kernelWidth:(NSUInteger) kernelWidth
            kernelHeight:(NSUInteger) kernelHeight
            strideInPixelsX:(NSUInteger) strideInPixelsX
            strideInPixelsY:(NSUInteger) strideInPixelsY
            dilationFactorX:(NSUInteger) dilationFactorX
            dilationFactorY:(NSUInteger) dilationFactorY
```

Init a node representing a MPSCNNPooling kernel

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter
kernelWidth	The width of the max filter window
kernelHeight	The height of the max filter window
strideInPixelsX	The output stride (downsampling factor) in the x dimension.
strideInPixelsY	The output stride (downsampling factor) in the y dimension.
dilationFactorX	The dilation factor in the x dimension.
dilationFactorY	The dilation factor in the y dimension.

Returns

A new MPSNNFilter node for a MPSCNNPooling kernel.

5.19.2.5 initWithSource:kernelWidth:kernelHeight:strideInPixelsX:strideInPixelsY:dilationRateX:dilationRateY:()

```
- (nonnull instancetype) initWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            kernelWidth:(NSUInteger) kernelWidth
            kernelHeight:(NSUInteger) kernelHeight
            strideInPixelsX:(NSUInteger) strideInPixelsX
            strideInPixelsY:(NSUInteger) strideInPixelsY
            dilationRateX:(NSUInteger) dilationRateX
            dilationRateY:(NSUInteger) dilationRateY
```

Init a node representing a MPSCNNPooling kernel

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter
kernelWidth	The width of the max filter window
kernelHeight	The height of the max filter window
strideInPixelsX	The output stride (downsampling factor) in the x dimension.
strideInPixelsY	The output stride (downsampling factor) in the y dimension.
dilationRateX	The dilation factor in the x dimension.
dilationRateY	The dilation factor in the y dimension.

Returns

A new MPSNNFilter node for a MPSCNNPooling kernel.

5.19.2.6 nodeWithSource:filterSize:()

```
+ (nonnull instancetype) nodeWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            filterSize:(NSUInteger) size
```

Convenience initializer for MPSCNNDilatedPooling nodes with square non-overlapping kernels

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter
size	kernelWidth = kernelHeight = strideInPixelsX = strideInPixelsY = dilationFactorX = dilationFactorY = size

Returns

A new MPSNNFilter node for a MPSCNNDilatedPooling kernel.

5.19.3 Property Documentation

5.19.3.1 dilationFactorX

```
- (NSUInteger) dilationFactorX  [read], [nonatomic], [assign]
```

5.19.3.2 dilationFactorY

```
- (NSUInteger NSUInteger) dilationFactorY  [read], [nonatomic], [assign]
```

5.19.3.3 dilationRateX

```
- (NSUInteger NSUInteger NSUInteger) dilationRateX  [read], [nonatomic], [assign]
```

5.19.3.4 dilationRateY

```
- (NSUInteger) dilationRateY  [read], [nonatomic], [assign]
```

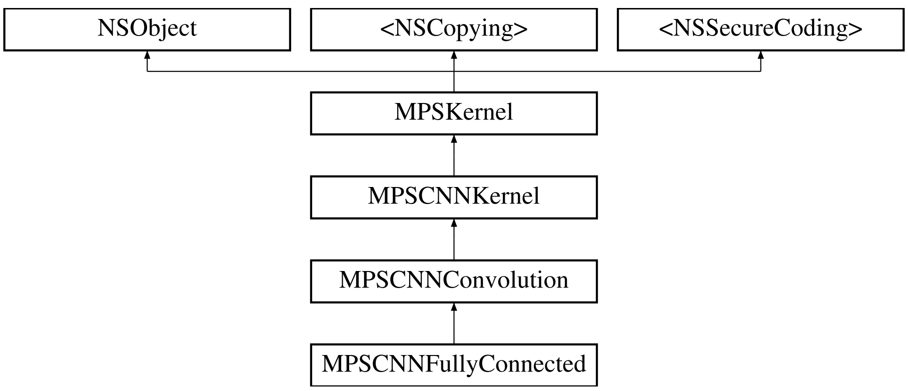
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.20 MPSCNNFullyConnected Class Reference

```
#import <MPSCNNConvolution.h>
```

Inheritance diagram for MPSCNNFullyConnected:



Instance Methods

- (nonnull instancetype) - initWithDevice:convolutionDescriptor:kernelWeights:biasTerms:flags:
- (nonnull instancetype) - initWithDevice:weights:
- (nullable instancetype) - initWithCoder:device:
- (nonnull instancetype) - initWithDevice:

Additional Inherited Members

5.20.1 Detailed Description

This depends on Metal.framework The MPSCNNFullyConnected specifies a fully connected convolution layer a.k.a. Inner product layer. A fully connected CNN layer is one where every input channel is connected to every output channel. The kernel width is equal to width of source image and the kernel height is equal to the height of source image. Width and height of the output is 1x1. Thus, it takes a srcW x srcH x Ni MPSCNNImage, convolves it with Weights[No][SrcW][srcH][Ni] and produces a 1 x 1 x No output. The following must be true:

```
kernelWidth  == source.width
kernelHeight == source.height
clipRect.size.width == 1
clipRect.size.height == 1
```

One can think of a fully connected layer as a matrix multiplication that flattens an image into a vector of length srcW∗srcH∗Ni. The weights are arragned in a matrix of dimension No x (srcW∗srcH∗Ni) for product output vectors of length No. The strideInPixelsX, strideInPixelsY, and group must be 1. Offset is not applicable and is ignored. Since clipRect is clamped to the destination image bounds, if the destination is 1x1, one doesn't need to set the clipRect.

Note that one can implement an inner product using MPSCNNConvolution by setting

```
offset = (kernelWidth/2,kernelHeight/2)
clipRect.origin = (ox,oy), clipRect.size = (1,1)
strideX = strideY = group = 1
```

However, using the MPSCNNFullyConnected for this is better for performance as it lets us choose the most performant method which may not be possible when using a general convolution. For example, we may internally use matrix multiplication or special reduction kernels for a specific platform.

5.20.2 Method Documentation

5.20.2.1 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSCNNConvolution.

5.20.2.2 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

A pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSCNNConvolution.

5.20.2.3 initWithDevice:convolutionDescriptor:kernelWeights:biasTerms:flags:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            convolutionDescriptor:(const MPSCNNConvolutionDescriptor *__nonnull) fullyConnected↵
Descriptor
            kernelWeights:(const float *__nonnull) kernelWeights
            biasTerms:(const float *__nullable) biasTerms
            flags:(MPSCNNConvolutionFlags) flags
```

Initializes a fully connected kernel.

Parameters

device	The MTLDevice on which this MPSCNNFullyConnected filter will be used
fullyConnectedDescriptor	A pointer to a MPSCNNConvolutionDescriptor. strideInPixelsX, strideInPixelsY and group properties of fullyConnectedDescriptor must be set to 1 (default).

Parameters

kernelWeights	A pointer to a weights array. Each entry is a float value. The number of entries is = inputFeatureChannels ∗ outputFeatureChannels ∗ kernelHeight ∗ kernelWidth The layout of filter weight is so that it can be reinterpreted as 4D tensor (array) weight[outputChannels][kernelHeight][kernelWidth][inputChannels / groups] Weights are converted to half float (fp16) internally for best performance.
biasTerms	A pointer to bias terms to be applied to the convolution output. Each entry is a float value. The number of entries is = numberOfOutputFeatureMaps
flags	Currently unused. Pass MPSCNNConvolutionFlagsNone

Returns

A valid MPSCNNConvolution object or nil, if failure.

Reimplemented from MPSCNNConvolution.

5.20.2.4 initWithDevice:weights:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            weights:(nonnull id< MPSCNNConvolutionDataSource >) weights
```

Initializes a fully connected kernel

Parameters

device	The MTLDevice on which this MPSCNNFullyConnected filter will be used
weights	A pointer to a object that conforms to the MPSCNNConvolutionDataSource protocol. The MPSCNNConvolutionDataSource protocol declares the methods that an instance of MPSCNNFullyConnected uses to obtain the weights and bias terms for the CNN fully connected filter.

Returns

A valid MPSCNNFullyConnected object or nil, if failure.

Reimplemented from MPSCNNConvolution.

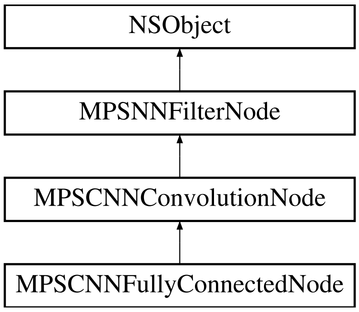
The documentation for this class was generated from the following file:

- MPSCNNConvolution.h

5.21 MPSCNNFullyConnectedNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSCNNFullyConnectedNode:



Instance Methods

- (nonnull instancetype) - initWithSource:weights:

Class Methods

- (nonnull instancetype) + nodeWithSource:weights:

Additional Inherited Members

5.21.1 Detailed Description

A MPSNNFilterNode representing a MPSCNNFullyConnected kernel

5.21.2 Method Documentation

5.21.2.1 initWithSource:weights:()

```
- (nonnull instancetype) initWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            weights:(nonnull id< MPSCNNConvolutionDataSource >) weights
```

Init a node representing a MPSCNNFullyConnected kernel

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter
weights	A pointer to a valid object conforming to the MPSCNNConvolutionDataSource protocol. This object is provided by you to encapsulate storage for convolution weights and biases.

Returns

A new MPSNNFilter node for a MPSCNNFullyConnected kernel.

Implements MPSCNNConvolutionNode.

5.21.2.2 nodeWithSource:weights:()

```
+ (nonnull instancetype) nodeWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            weights:(nonnull id< MPSCNNConvolutionDataSource >) weights
```

Init an autoreleased not representing a MPSCNNFullyConnected kernel

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter
weights	A pointer to a valid object conforming to the MPSCNNConvolutionDataSource protocol. This object is provided by you to encapsulate storage for convolution weights and biases.

Returns

A new MPSNNFilter node for a MPSCNNConvolution kernel.

Implements MPSCNNConvolutionNode.

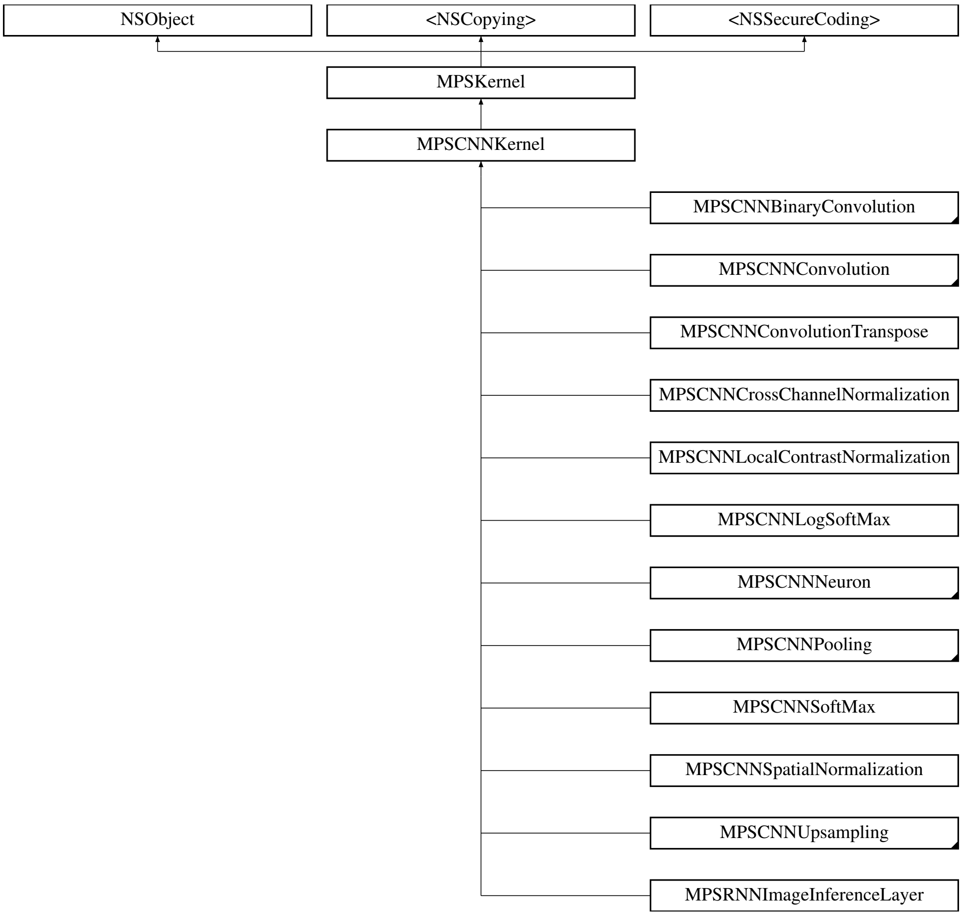
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.22 MPSCNNKernel Class Reference

```
#import <MPSCNNKernel.h>
```

Inheritance diagram for MPSCNNKernel:



Instance Methods

- (nonnull instancetype) - initWithDevice:
- (nullable instancetype) - initWithCoder:device:
- (void) - encodeToCommandBuffer:sourceImage:destinationImage:
- (MPSImage ∗__nonnull) - encodeToCommandBuffer:sourceImage:

Properties

- MPSOffset offset
- MTLRegion clipRect
- NSUInteger destinationFeatureChannelOffset
- MPSImageEdgeMode edgeMode
- NSUInteger kernelWidth
- NSUInteger kernelHeight
- NSUInteger strideInPixelsX
- NSUInteger strideInPixelsY
- BOOL isBackwards
- id< MPSNNPadding > padding
- id< MPSNNPadding > id< MPSImageAllocator > destinationImageAllocator

Additional Inherited Members

5.22.1 Detailed Description

This depends on Metal.framework Describes a convolution neural network kernel. A MPSCNNKernel consumes one MPSImage and produces one MPSImage.

```
The region overwritten in the destination MPSImage is described
by the clipRect.  The top left corner of the region consumed (ignoring
adjustments for filter size -- e.g. convolution filter size) is given
by the offset. The size of the region consumed is a function of the
clipRect size and any subsampling caused by pixel strides at work,
e.g. MPSCNNPooling.strideInPixelsX/Y.  Where the offset + clipRect
would cause a {x,y} pixel address not in the image to be read, the
edgeMode is used to determine what value to read there.

The Z/depth component of the offset, clipRect.origin and clipRect.size
indexes which images to use. If the MPSImage contains only a single image
then these should be offset.z = 0, clipRect.origin.z = 0
and clipRect.size.depth = 1. If the MPSImage contains multiple images,
clipRect.size.depth refers to number of images to process. Both source
and destination MPSImages must have at least this many images. offset.z
refers to starting source image index. Thus offset.z + clipRect.size.depth must
be <= source.numberOfImages. Similarly, clipRect.origin.z refers to starting
image index in destination. So clipRect.origin.z + clipRect.size.depth must be
<= destination.numberOfImage.

destinationFeatureChannelOffset property can be used to control where the MPSKernel will
start writing in feature channel dimension. For example, if the destination image has
64 channels, and MPSKernel outputs 32 channels, by default channels 0-31 of destination
will be populated by MPSKernel. But if we want this MPSKernel to populate channel 32-63
of the destination, we can set destinationFeatureChannelOffset = 32.
A good example of this is concat (concatenation) operation in Tensor Flow. Suppose
we have a src = w x h x Ni which goes through CNNConvolution_0 which produces
output O0 = w x h x N0 and CNNConvolution_1 which produces output O1 = w x h x N1 followed
by concatenation which produces O = w x h x (N0 + N1). We can achieve this by creating
an MPSImage with dimensions O = w x h x (N0 + N1) and using this as destination of
both convolutions as follows
    CNNConvolution0: destinationFeatureChannelOffset = 0, this will output N0 channels starting at
                     channel 0 of destination thus populating [0,N0-1] channels.
    CNNConvolution1: destinationFeatureChannelOffset = N0, this will output N1 channels starting at
                     channel N0 of destination thus populating [N0,N0+N1-1] channels.


A MPSCNNKernel can be saved to disk / network using NSCoders such as NSKeyedArchiver.
When decoding, the system default MTLDevice will be chosen unless the NSCoder adopts
the <MPSDeviceProvider> protocol.  To accomplish this you will likely need to subclass your
unarchiver to add this method.
```

5.22.2 Method Documentation

5.22.2.1 encodeToCommandBuffer:sourceImage:()

```
- (MPSImage * __nonnull) encodeToCommandBuffer:
            (nonnull id< MTLCommandBuffer >) commandBuffer
            sourceImage:(MPSImage *__nonnull) sourceImage
```

Encode a MPSCNNKernel into a command Buffer. Create a texture to hold the result and return it. In the first iteration on this method, encodeToCommandBuffer:sourceImage:destinationImage: some work was left for the developer to do in the form of correctly setting the offset property and sizing the result buffer. With the introduction of the padding policy (see padding property) the filter can do this work itself. If you would like to have some input into what sort of MPSImage (e.g. temporary vs. regular) or what size it is or where it is allocated, you may set the destinationImageAllocator to allocate the image yourself.

This method uses the MPSNNPadding padding property to figure out how to size the result image and to set the offset property. See discussion in MPSNeuralNetworkTypes.h.

Parameters

commandBuffer	The command buffer
sourceImage	A MPSImage to use as the source images for the filter.

Returns

A MPSImage or MPSTemporaryImage allocated per the destinationImageAllocator containing the output of the graph. The offset property will be adjusted to reflect the offset used during the encode. The returned image will be automatically released when the command buffer completes. If you want to keep it around for longer, retain the image. (ARC will do this for you if you use it later.)

5.22.2.2 encodeToCommandBuffer:sourceImage:destinationImage:()

```
- (void) encodeToCommandBuffer:
            (nonnull id< MTLCommandBuffer >) commandBuffer
            sourceImage:(MPSImage *__nonnull) sourceImage
            destinationImage:(MPSImage *__nonnull) destinationImage
```

Encode a MPSCNNKernel into a command Buffer. The operation shall proceed out-of-place. This is the older style of encode which reads the offset, doesn't change it, and ignores the padding method.

Parameters

commandBuffer	A valid MTLCommandBuffer to receive the encoded filter
sourceImage	A valid MPSImage object containing the source image.
destinationImage	A valid MPSImage to be overwritten by result image. destinationImage may not alias sourceImage.

5.22.2.3 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSKernel.

Reimplemented in MPSCNNBinaryConvolution, MPSCNNBinaryFullyConnected, MPSCNNConvolution↵Transpose, MPSCNNConvolution, MPSCNNFullyConnected, MPSRNNImageInferenceLayer, MPSCNNNeuron, MPSCNNDilatedPoolingMax, MPSCNNPoolingAverage, MPSCNNPoolingL2Norm, MPSCNNCrossChannel↵Normalization, MPSCNNPooling, MPSCNNPoolingMax, MPSCNNLocalContrastNormalization, and MPSCNN↵SpatialNormalization.

5.22.2.4 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

A pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSKernel.

Reimplemented in MPSCNNBinaryConvolution, MPSCNNBinaryFullyConnected, MPSCNNConvolutionTranspose, MPSCNNConvolution, MPSCNNFullyConnected, MPSRNNImageInferenceLayer, MPSCNNNeuronELU, MPSC↵NNCrossChannelNormalization, MPSCNNPooling, MPSCNNNeuronSoftPlus, MPSCNNNeuronSoftSign, MPSC↵NNNeuronTanH, MPSCNNNeuronAbsolute, MPSCNNLocalContrastNormalization, MPSCNNNeuronHardSigmoid, MPSCNNNeuronReLU, MPSCNNNeuronSigmoid, MPSCNNNeuronLinear, MPSCNNSpatialNormalization, and MPSCNNUpsampling.

5.22.3 Property Documentation

5.22.3.1 clipRect

```
- clipRect  [read], [write], [nonatomic], [assign]
```

An optional clip rectangle to use when writing data. Only the pixels in the rectangle will be overwritten. A MTLRegion that indicates which part of the destination to overwrite. If the clipRect does not lie completely within the destination image, the intersection between clip rectangle and destination bounds is used. Default: MPSRectNoClip (MPS↵Kernel::MPSRectNoClip) indicating the entire image. clipRect.origin.z is the index of starting destination image in batch processing mode. clipRect.size.depth is the number of images to process in batch processing mode.

See Also: MetalPerformanceShaders.h subsubsection_clipRect

5.22.3.2 destinationFeatureChannelOffset

```
- destinationFeatureChannelOffset [read], [write], [nonatomic], [assign]
```

The number of channels in the destination MPSImage to skip before writing output. This is the starting offset into the destination image in the feature channel dimension at which destination data is written. This allows an application to pass a subset of all the channels in MPSImage as output of MPSKernel. E.g. Suppose MPSImage has 24 channels and a MPSKernel outputs 8 channels. If we want channels 8 to 15 of this MPSImage to be used as output, we can set destinationFeatureChannelOffset = 8. Note that this offset applies independently to each image when the MPSImage is a container for multiple images and the MPSCNNKernel is processing multiple images (clipRect.size.depth > 1). The default value is 0 and any value specifed shall be a multiple of 4. If MPSKernel outputs N channels, destination image MUST have at least destinationFeatureChannelOffset + N channels. Using a destination image with insufficient number of feature channels result in an error. E.g. if the MPSCNNConvolution outputs 32 channels, and destination has 64 channels, then it is an error to set destinationFeatureChannelOffset > 32.

5.22.3.3 destinationImageAllocator

```
- (id<MPSNNPadding> id<MPSImageAllocator>) destinationImageAllocator [read], [write], [nonatomic],
[retain]
```

Method to allocate the result image for -encodeToCommandBuffer:sourceImage: Default: defaultAllocator (MPS↵ TemporaryImage)

5.22.3.4 edgeMode

```
- edgeMode [read], [write], [nonatomic], [assign]
```

The MPSImageEdgeMode to use when texture reads stray off the edge of an image Most MPSKernel objects can read off the edge of the source image. This can happen because of a negative offset property, because the offset + clipRect.size is larger than the source image or because the filter looks at neighboring pixels, such as a Convolution filter. Default: MPSImageEdgeModeZero.

See Also: MetalPerformanceShaders.h subsubsection_edgemode Note: For MPSCNNPoolingAverage specifying edge mode MPSImageEdgeModeClamp is interpreted as a "shrink-to-edge" operation, which shrinks the effective filtering window to remain within the source image borders.

5.22.3.5 isBackwards

```
- isBackwards [read], [nonatomic], [assign]
```

YES if the filter operates backwards. This influences how strideInPixelsX/Y should be interpreted. Most filters either have stride 1 or are reducing, meaning that the result image is smaller than the original by roughly a factor of the stride. A few "backward" filters (e.g unpooling) are intended to "undo" the effects of an earlier forward filter, and so enlarge the image. The stride is in the destination coordinate frame rather than the source coordinate frame.

5.22.3.6 kernelHeight

```
- kernelHeight [read], [nonatomic], [assign]
```

The height of the MPSCNNKernel filter window This is the vertical diameter of the region read by the filter for each result pixel. If the MPSCNNKernel does not have a filter window, then 1 will be returned.

5.22.3.7 kernelWidth

```
- kernelWidth [read], [nonatomic], [assign]
```

The width of the MPSCNNKernel filter window This is the horizontal diameter of the region read by the filter for each result pixel. If the MPSCNNKernel does not have a filter window, then 1 will be returned.

5.22.3.8 offset

```
- offset [read], [write], [nonatomic], [assign]
```

The position of the destination clip rectangle origin relative to the source buffer. The offset is defined to be the position of clipRect.origin in source coordinates. Default: {0,0,0}, indicating that the top left corners of the clipRect and source image align. offset.z is the index of starting source image in batch processing mode.

See Also: MetalPerformanceShaders.h subsubsection_mpsoffset

5.22.3.9 padding

```
- padding [read], [write], [nonatomic], [assign]
```

The padding method used by the filter This influences how the destination image is sized and how the offset into the source image is set. It is used by the -encode methods that return a MPSImage from the left hand side.

5.22.3.10 strideInPixelsX

```
- strideInPixelsX [read], [nonatomic], [assign]
```

The downsampling (or upsampling if a backwards filter) factor in the horizontal dimension If the filter does not do up or downsampling, 1 is returned.

5.22.3.11 strideInPixelsY

```
- strideInPixelsY [read], [nonatomic], [assign]
```

The downsampling (or upsampling if a backwards filter) factor in the vertical dimension If the filter does not do up or downsampling, 1 is returned.

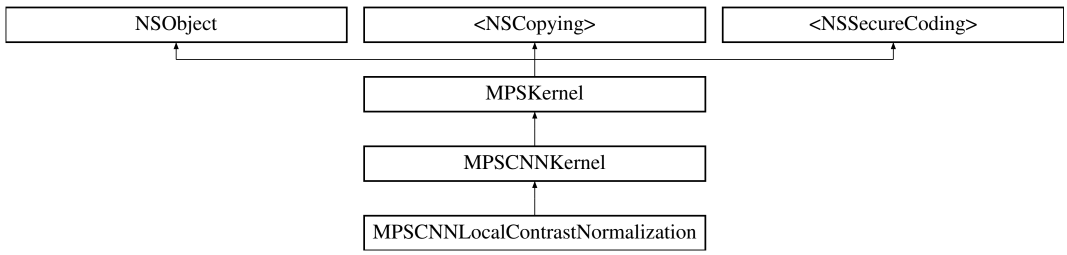
The documentation for this class was generated from the following file:

- MPSCNNKernel.h

5.23 MPSCNNLocalContrastNormalization Class Reference

```
#import <MPSCNNNormalization.h>
```

Inheritance diagram for MPSCNNLocalContrastNormalization:



Instance Methods

- (nonnull instancetype) - initWithDevice:kernelWidth:kernelHeight:
- (nullable instancetype) - initWithCoder:device:
- (nonnull instancetype) - initWithDevice:

Properties

- float alpha
- float beta
- float delta
- float p0
- float pm
- float ps
- NSUInteger kernelWidth
- NSUInteger kernelHeight

Additional Inherited Members

5.23.1 Detailed Description

This depends on Metal.framework Specifies the local contrast normalization filter. The local contrast normalization is quite similar to spatial normalization (see MPSCNNSpatialNormalization) in that it applies the filter over local regions which extend spatially, but are in separate feature channels (i.e., they have shape 1 x kernelWidth x kernelHeight), but instead of dividing by the local "energy" of the feature, the denominator uses the local variance of the feature - effectively the mean value of the feature is subtracted from the signal. For each feature channel, the function computes the variance VAR(i,j) and mean M(i,j) of X(i,j) inside each rectangle around the spatial point (i,j).

Then the result is computed for each element of X as follows:

```
Y(i,j) = pm + ps * ( X(i,j) - p0 * M(i,j)) / (delta + alpha * VAR(i,j))^beta,
```

where kw and kh are the kernelWidth and the kernelHeight and pm, ps and p0 are parameters that can be used to offset and scale the result in various ways. For example setting pm=0, ps=1, p0=1, delta=0, alpha=1.0 and beta=0.5 scales input data so that the result has unit variance and zero mean, provided that input variance is positive. It is the end-users responsibility to ensure that the combination of the parameters delta and alpha does not result in a situation where the denominator becomes zero - in such situations the resulting pixel-value is undefined. A good way to guard against tiny variances is to regulate the expression with a small value for delta, for example delta = 1/1024 = 0.0009765625.

5.23.2 Method Documentation

5.23.2.1 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSCNNKernel.

5.23.2.2 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

A pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSCNNKernel.

5.23.2.3 initWithDevice:kernelWidth:kernelHeight:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            kernelWidth:(NSUInteger) kernelWidth
            kernelHeight:(NSUInteger) kernelHeight
```

Initialize a local contrast normalization filter

Parameters

device	The device the filter will run on
kernelWidth	The width of the kernel
kernelHeight	The height of the kernel

Returns

A valid MPSCNNLocalContrastNormalization object or nil, if failure.

NOTE: For now, kernelWidth must be equal to kernelHeight

5.23.3 Property Documentation

5.23.3.1 alpha

```
- alpha [read], [write], [nonatomic], [assign]
```

The value of alpha. Default is 1.0

5.23.3.2 beta

```
- beta [read], [write], [nonatomic], [assign]
```

The value of beta. Default is 0.5

5.23.3.3 delta

```
- delta [read], [write], [nonatomic], [assign]
```

The value of delta. Default is 1/1024

5.23.3.4 kernelHeight

```
- kernelHeight [read], [nonatomic], [assign]
```

The height of the filter window

5.23.3.5 kernelWidth

```
- kernelWidth [read], [nonatomic], [assign]
```

The width of the filter window

5.23.3.6 p0

```
- p0 [read], [write], [nonatomic], [assign]
```

The value of p0. Default is 1.0

5.23.3.7 pm

```
- pm [read], [write], [nonatomic], [assign]
```

The value of pm. Default is 0.0

5.23.3.8 ps

```
- ps [read], [write], [nonatomic], [assign]
```

The value of ps. Default is 1.0

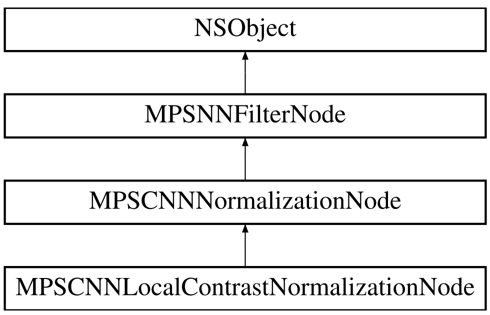
The documentation for this class was generated from the following file:

- MPSCNNNormalization.h

5.24 MPSCNNLocalContrastNormalizationNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSCNNLocalContrastNormalizationNode:



Instance Methods

- (nonnull instancetype) - initWithSource:kernelSize:
- (nonnull instancetype) - initWithSource:

Class Methods

- (nonnull instancetype) + nodeWithSource:kernelSize:

Properties

- float pm
- float ps
- float p0
- NSUInteger kernelWidth
- NSUInteger kernelHeight

5.24.1 Method Documentation

5.24.1.1 initWithSource:()

```
- (nonnull instancetype) initWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
```

Implements MPSCNNNormalizationNode.

5.24.1.2 initWithSource:kernelSize:()

```
- (nonnull instancetype) initWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            kernelSize:(NSUInteger) kernelSize
```

5.24.1.3 nodeWithSource:kernelSize:()

```
+ (nonnull instancetype) nodeWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            kernelSize:(NSUInteger) kernelSize
```

5.24.2 Property Documentation

5.24.2.1 kernelHeight

```
- (NSUInteger) kernelHeight [read], [write], [nonatomic], [assign]
```

5.24.2.2 kernelWidth

```
- (NSUInteger) kernelWidth [read], [write], [nonatomic], [assign]
```

5.24.2.3 p0

```
- (float) p0 [read], [write], [nonatomic], [assign]
```

5.24.2.4 pm

```
- (float) pm [read], [write], [nonatomic], [assign]
```

5.24.2.5 ps

```
- (float) ps [read], [write], [nonatomic], [assign]
```

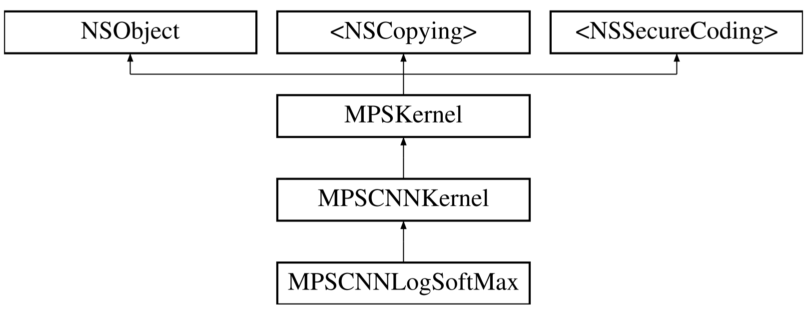
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.25 MPSCNNLogSoftMax Class Reference

```
#import <MPSCNNSoftMax.h>
```

Inheritance diagram for MPSCNNLogSoftMax:



Additional Inherited Members

5.25.1 Detailed Description

This depends on Metal.framework The logarithmic softmax filter can be achieved by taking the natural logarithm of the the result of the softmax filter. The results are often used to construct a loss function to be minimized when training neural networks. For each feature channel per pixel in an image in a feature map, the logarithmic softmax filter computes the following: result channel in pixel = pixel(x,y,k)) - ln{sum(exp(pixel(x,y,0)) ... exp(pixel(x,y,N-1))} where N is the number of feature channels and y = ln{x} satisfies e^y = x.

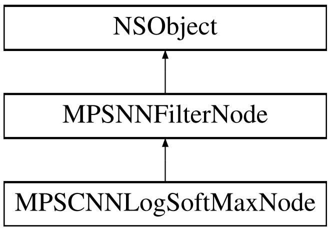
The documentation for this class was generated from the following file:

- MPSCNNSoftMax.h

5.26 MPSCNNLogSoftMaxNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSCNNLogSoftMaxNode:



Instance Methods

- (nonnull instancetype) - initWithSource:

Class Methods

- (nonnull instancetype) + nodeWithSource:

Additional Inherited Members

5.26.1 Detailed Description

Node representing a MPSCNNLogSoftMax kernel

5.26.2 Method Documentation

5.26.2.1 initWithSource:()

```
- (nonnull instancetype) initWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
```

Init a node representing a MPSCNNLogSoftMax kernel

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter

Returns

A new MPSNNFilter node for a MPSCNNLogSoftMax kernel.

5.26.2.2 nodeWithSource:()

```
+ (nonnull instancetype) nodeWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
```

Init a node representing a autoreleased MPSCNNLogSoftMax kernel

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter

Returns

A new MPSNNFilter node for a MPSCNNLogSoftMax kernel.

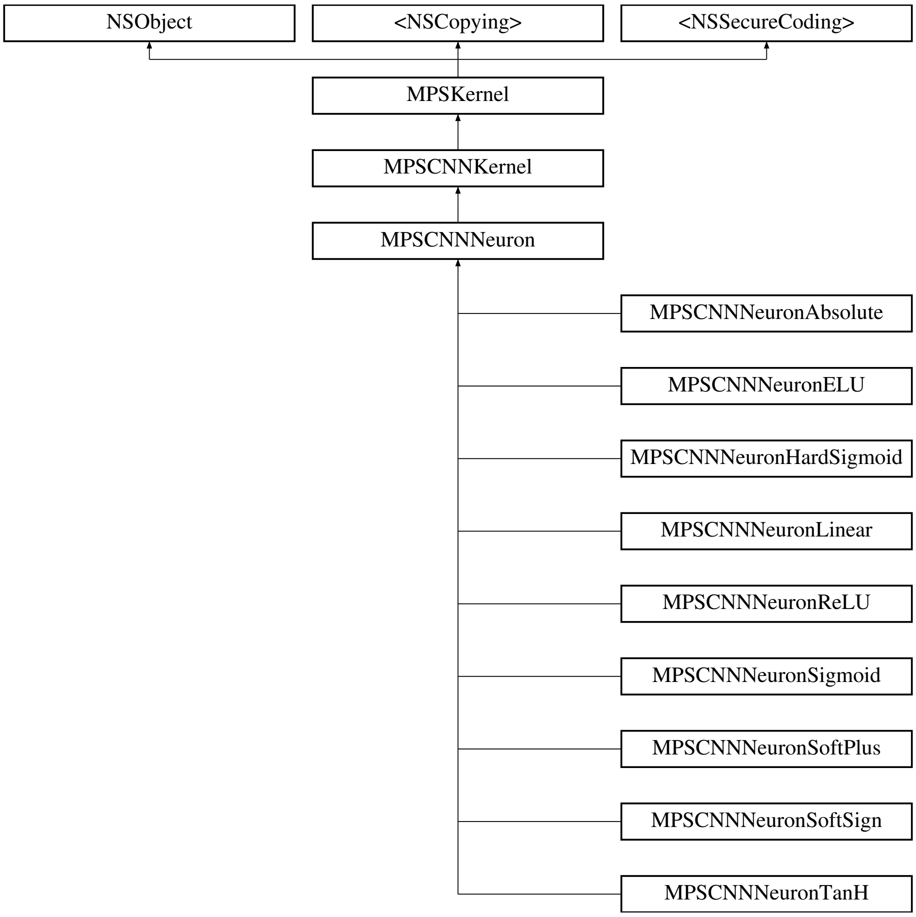
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.27 MPSCNNNeuron Class Reference

```
#import <MPSCNNConvolution.h>
```

Inheritance diagram for MPSCNNNeuron:



Instance Methods

- (nullable instancetype) - initWithCoder:device:

Additional Inherited Members

5.27.1 Detailed Description

This depends on Metal.framework This filter applies a neuron activation function. You must use one of the sub-classes of MPSCNNNeuron

5.27.2 Method Documentation

5.27.2.1 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
                (NSCoder *__nonnull) aDecoder
                device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSCNNKernel.

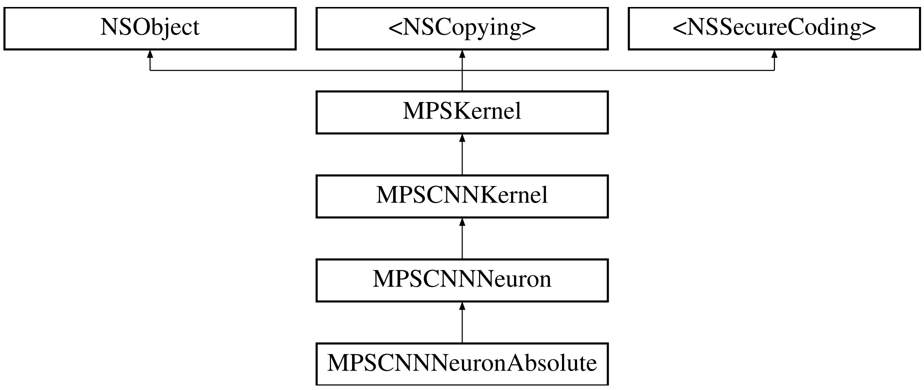
The documentation for this class was generated from the following file:

- MPSCNNConvolution.h

5.28 MPSCNNNeuronAbsolute Class Reference

```
#import <MPSCNNConvolution.h>
```

Inheritance diagram for MPSCNNNeuronAbsolute:



Instance Methods

- (nonnull instancetype) - initWithDevice:

Additional Inherited Members

5.28.1 Detailed Description

This depends on Metal.framework Specifies the absolute neuron filter. For each pixel, applies the following function: f(x) = | x |

5.28.2 Method Documentation

5.28.2.1 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Initialize a neuron filter

Parameters

device	The device the filter will run on

Returns

A valid MPSCNNNeuronAbsolute object or nil, if failure.

Reimplemented from MPSCNNKernel.

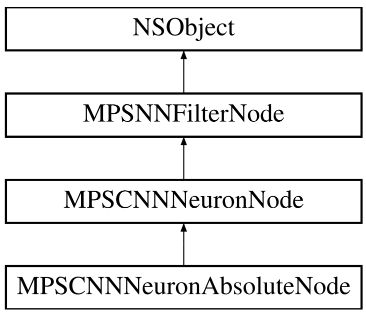
The documentation for this class was generated from the following file:

- MPSCNNConvolution.h

5.29 MPSCNNNeuronAbsoluteNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSCNNNeuronAbsoluteNode:



Class Methods

- (nonnull instancetype) + nodeWithSource:

Additional Inherited Members

5.29.1 Detailed Description

A node representing a MPSCNNNeuronAbsolute kernel For each pixel, applies the following function:

```
f(x) = fabs(x)
```

5.29.2 Method Documentation

5.29.2.1 nodeWithSource:()

```
+ (nonnull instancetype) nodeWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
```

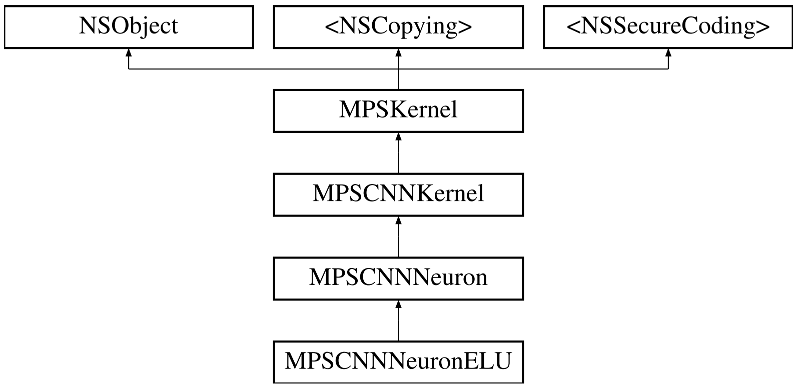
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.30 MPSCNNNeuronELU Class Reference

```
#import <MPSCNNConvolution.h>
```

Inheritance diagram for MPSCNNNeuronELU:



Instance Methods

- (nonnull instancetype) - initWithDevice:a:
- (nonnull instancetype) - initWithDevice:

Properties

- float a

Additional Inherited Members

5.30.1 Detailed Description

This depends on Metal.framework Specifies the parametric ELU neuron filter. For each pixel, applies the following function: $f(x) = [\ a * (exp(x) - 1), x < 0\ [\ x\ , x >= 0$

5.30.2 Method Documentation

5.30.2.1 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

A pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSCNNKernel.

5.30.2.2 initWithDevice:a:()

```
- (nonnull instancetype) initWithDevice:
             (nonnull id< MTLDevice >) device
             a:(float) a
```

Initialize a parametric ELU neuron filter

Parameters

device	The device the filter will run on
a	Filter property "a". See class discussion.

Returns

A valid MPSCNNNeuronELU object or nil, if failure.

5.30.3 Property Documentation

5.30.3.1 a

```
- (float) a  [read], [atomic], [assign]
```

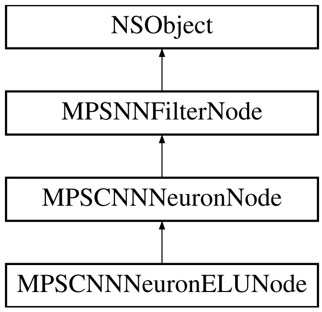
The documentation for this class was generated from the following file:

- MPSCNNConvolution.h

5.31 MPSCNNNeuronELUNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSCNNNeuronELUNode:



Class Methods

- (nonnull instancetype) + nodeWithSource:a:

Additional Inherited Members

5.31.1 Detailed Description

A node representing a MPSCNNNeuronELU kernel For each pixel, applies the following function:

```
f(x) = a * exp(x) - 1, x <  0
       x             , x >= 0
```

5.31.2 Method Documentation

5.31.2.1 nodeWithSource:a:()

```
+ (nonnull instancetype) nodeWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            a:(float) a
```

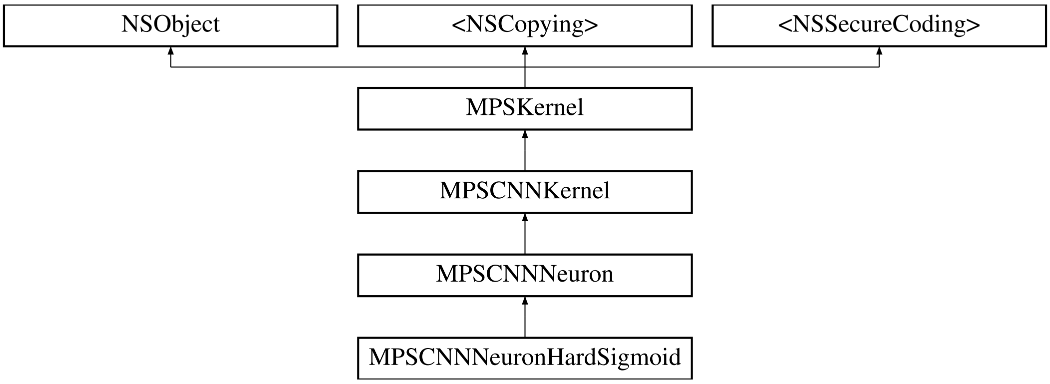
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.32 MPSCNNNeuronHardSigmoid Class Reference

```
#import <MPSCNNConvolution.h>
```

Inheritance diagram for MPSCNNNeuronHardSigmoid:



Instance Methods

- (nonnull instancetype) - initWithDevice:a:b:
- (nonnull instancetype) - initWithDevice:

Properties

- float a
- float b

Additional Inherited Members

5.32.1 Detailed Description

This depends on Metal.framework Specifies the hard sigmoid neuron filter. For each pixel, applies the following function: f(x) = clamp((a ∗ x) + b, 0, 1)

5.32.2 Method Documentation

5.32.2.1 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

A pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSCNNKernel.

5.32.2.2 initWithDevice:a:b:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            a:(float) a
            b:(float) b
```

Initialize a neuron filter

Parameters

device	The device the filter will run on
a	Filter property "a". See class discussion.
b	Filter property "b". See class discussion.

Returns

A valid MPSCNNNeuronHardSigmoid object or nil, if failure.

5.32.3 Property Documentation

5.32.3.1 a

```
- (float) a  [read], [atomic], [assign]
```

5.32.3.2 b

```
- (float) b  [read], [atomic], [assign]
```

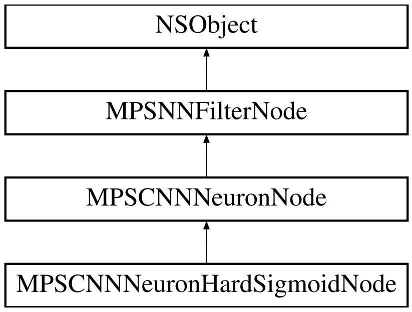
The documentation for this class was generated from the following file:

- MPSCNNConvolution.h

5.33 MPSCNNNeuronHardSigmoidNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSCNNNeuronHardSigmoidNode:



Instance Methods

- (nonnull instancetype) - initWithSource:a:b:

Class Methods

- (nonnull instancetype) + nodeWithSource:a:b:

Additional Inherited Members

5.33.1 Detailed Description

A node representing a MPSCNNNeuronHardSigmoid kernel For each pixel, applies the following function:

```
f(x) = clamp((a * x) + b, 0, 1)
```

5.33.2 Method Documentation

5.33.2.1 initWithSource:a:b:()

```
- (nonnull instancetype) initWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            a:(float) a
            b:(float) b
```

Init a node representing a MPSCNNNeuronHardSigmoid kernel

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter
a	See discussion above.
b	See discussion above.

Returns

A new MPSNNFilter node for a MPSCNNNeuronHardSigmoid kernel.

5.33.2.2 nodeWithSource:a:b:()

```
+ (nonnull instancetype) nodeWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            a:(float) a
            b:(float) b
```

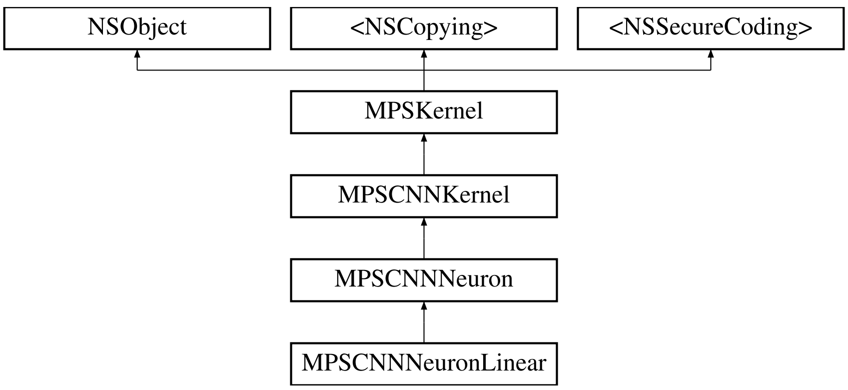
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.34 MPSCNNNeuronLinear Class Reference

```
#import <MPSCNNConvolution.h>
```

Inheritance diagram for MPSCNNNeuronLinear:



Instance Methods

- (nonnull instancetype) - initWithDevice:a:b:
- (nonnull instancetype) - initWithDevice:

Properties

- float a
- float b

Additional Inherited Members

5.34.1 Detailed Description

This depends on Metal.framework Specifies the linear neuron filter. For each pixel, applies the following function: f(x) = a ∗ x + b

5.34.2 Method Documentation

5.34.2.1 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

A pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSCNNKernel.

5.34.2.2 initWithDevice:a:b:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            a:(float) a
            b:(float) b
```

Initialize the linear neuron filter

Parameters

device	The device the filter will run on
a	Filter property "a". See class discussion.
b	Filter property "b". See class discussion.

Returns

A valid MPSCNNNeuronLinear object or nil, if failure.

5.34.3 Property Documentation

5.34.3.1 a

```
- (float) a [read], [atomic], [assign]
```

5.34.3.2 b

```
- (float) b [read], [atomic], [assign]
```

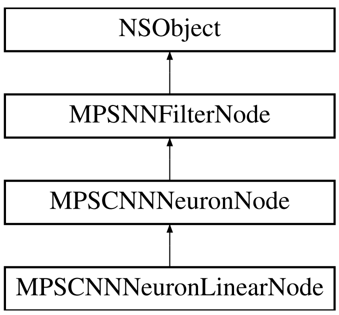
The documentation for this class was generated from the following file:

- MPSCNNConvolution.h

5.35 MPSCNNNeuronLinearNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSCNNNeuronLinearNode:



Instance Methods

- (nonnull instancetype) - initWithSource:a:b:

Class Methods

- (nonnull instancetype) + nodeWithSource:a:b:

Additional Inherited Members

5.35.1 Detailed Description

A node representing a MPSCNNNeuronLinear kernel For each pixel, applies the following function:

```
f(x) = a * x + b
```

5.35.2 Method Documentation

5.35.2.1 initWithSource:a:b:()

```
- (nonnull instancetype) initWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            a:(float) a
            b:(float) b
```

Init a node representing a MPSCNNNeuronLinear kernel

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter
a	See discussion above.
b	See discussion above.

Returns

A new MPSNNFilter node for a MPSCNNNeuronLinear kernel.

5.35.2.2 nodeWithSource:a:b:()

```
+ (nonnull instancetype) nodeWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
```

```
a:(float) a
b:(float) b
```

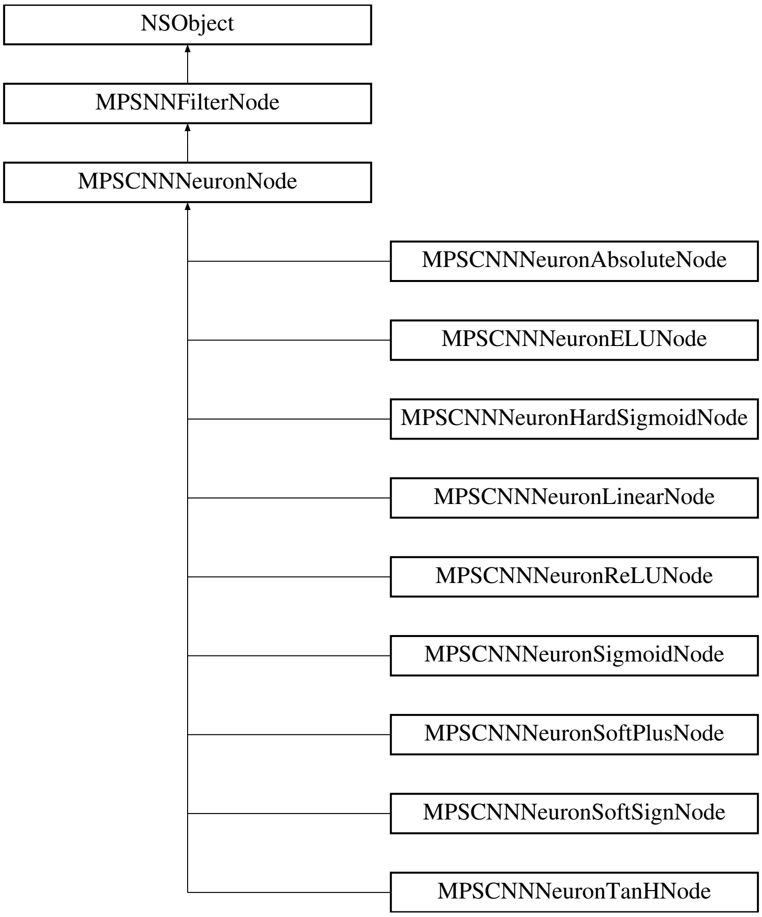
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.36 MPSCNNNeuronNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSCNNNeuronNode:



Instance Methods

- (MPSCNNNeuronNode ∗__nonnull) - initWithSource:type:a:b:
- (nonnull instancetype) - initWithSource:
- (nonnull instancetype) - initWithSource:a:

Class Methods

- (MPSCNNNeuronNode ∗__nonnull) + nodeWithSource:type:a:b:

Additional Inherited Members

5.36.1 Method Documentation

5.36.1.1 initWithSource:()

```
- (nonnull instancetype) initWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
```

Convenience method to init a node representing a MPSCNNNeuronELU kernel. Neuron parameters a and b will default to 1.0f

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter

Returns

A new MPSNNFilter node for a MPSCNNNeuronELU kernel.

5.36.1.2 initWithSource:a:()

```
- (nonnull instancetype) initWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            a:(float) a
```

Convenience method to init a node representing a MPSCNNNeuronELU kernel. Neuron parameter b (if it exists) will default to 1.0f

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter
a	See class discussion.

Returns

A new MPSNNFilter node for a MPSCNNNeuronELU kernel.

5.36.1.3 initWithSource:type:a:b:()

```
- (MPSCNNNeuronNode * __nonnull) initWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
```

```
            type:(MPSCNNNeuronType) type
            a:(float) a
            b:(float) b
```

Init a node representing a MPSCNNNeuron The class of the result depends on the contents of the descriptor.

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter
type	The type of neuron node to make
a	Parameter value a.
b	Parameter value b.

Returns

A new MPSCNNNeuronNode subclass instance for a MPSCNNNeuron kernel.

5.36.1.4 nodeWithSource:type:a:b:()

```
+ (MPSCNNNeuronNode * __nonnull) nodeWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            type:(MPSCNNNeuronType) type
            a:(float) a
            b:(float) b
```

Init a node representing a MPSCNNNeuron The class of the result depends on the contents of the descriptor.

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter
type	The type of neuron node to make
a	Parameter value a.
b	Parameter value b.

Returns

A new MPSCNNNeuronNode subclass instance for a MPSCNNNeuron kernel.

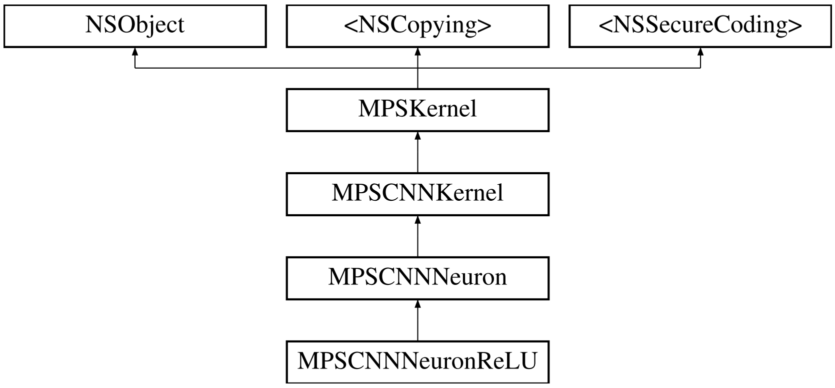
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.37 MPSCNNNeuronReLU Class Reference

```
#import <MPSCNNConvolution.h>
```

Inheritance diagram for MPSCNNNeuronReLU:



Instance Methods

- (nonnull instancetype) - initWithDevice:a:
- (nonnull instancetype) - initWithDevice:

Properties

- float a

Additional Inherited Members

5.37.1 Detailed Description

This depends on Metal.framework Specifies the ReLU neuron filter. For each pixel, applies the following function: f(x) = x, if x $>=$ 0 = a $*$ x if x $<$ 0 This is called Leaky ReLU in literature. Some literature defines classical ReLU as max(0, x). If you want this behavior, simply pass a = 0

5.37.2 Method Documentation

5.37.2.1 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

A pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSCNNKernel.

5.37.2.2 initWithDevice:a:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            a:(float) a
```

Initialize the ReLU neuron filter

Parameters

device	The device the filter will run on
a	Filter property "a". See class discussion.

Returns

A valid MPSCNNNeuronReLU object or nil, if failure.

5.37.3 Property Documentation

5.37.3.1 a

```
- (float) a [read], [atomic], [assign]
```

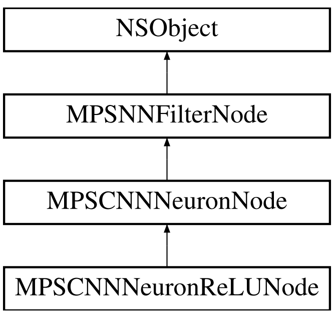
The documentation for this class was generated from the following file:

- MPSCNNConvolution.h

5.38 MPSCNNNeuronReLUNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSCNNNeuronReLUNode:



Class Methods

- (nonnull instancetype) + nodeWithSource:a:

Additional Inherited Members

5.38.1 Detailed Description

A node representing a MPSCNNNeuronReLU kernel For each pixel, applies the following function:

```
f(x) = x            if x >= 0
     = a * x        if x < 0
```

5.38.2 Method Documentation

5.38.2.1 nodeWithSource:a:()

```
+ (nonnull instancetype) nodeWithSource:
               (MPSNNImageNode *__nonnull) sourceNode
               a:(float) a
```

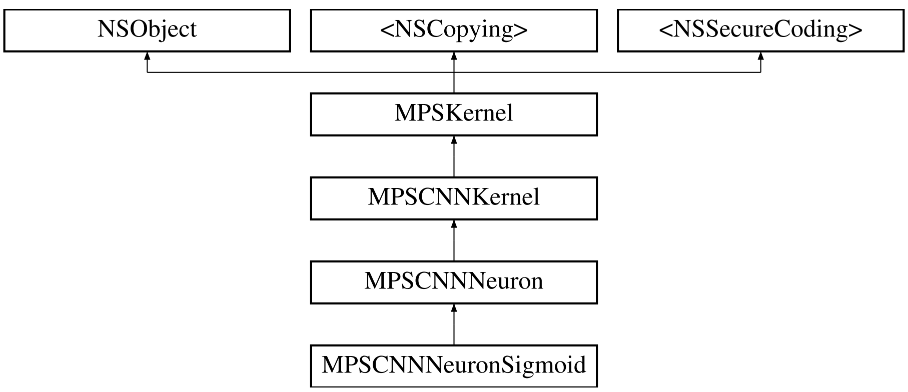
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.39 MPSCNNNeuronSigmoid Class Reference

```
#import <MPSCNNConvolution.h>
```

Inheritance diagram for MPSCNNNeuronSigmoid:



Instance Methods

- (nonnull instancetype) - initWithDevice:

Additional Inherited Members

5.39.1 Detailed Description

This depends on Metal.framework Specifies the sigmoid neuron filter. For each pixel, applies the following function: f(x) = 1 / (1 + e^-x)

5.39.2 Method Documentation

5.39.2.1 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Initialize a neuron filter

Parameters

device	The device the filter will run on

Returns

A valid MPSCNNNeuronSigmoid object or nil, if failure.

Reimplemented from MPSCNNKernel.

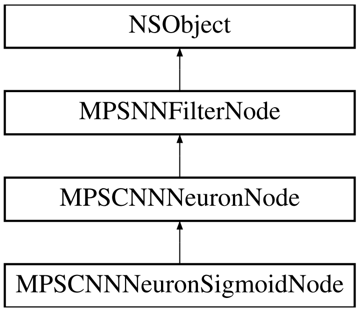
The documentation for this class was generated from the following file:

- MPSCNNConvolution.h

5.40 MPSCNNNeuronSigmoidNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSCNNNeuronSigmoidNode:



Class Methods

- (nonnull instancetype) + nodeWithSource:

Additional Inherited Members

5.40.1 Detailed Description

A node representing a MPSCNNNeuronSigmoid kernel For each pixel, applies the following function:

```
f(x) = 1 / (1 + e^-x)
```

5.40.2 Method Documentation

5.40.2.1 nodeWithSource:()

```
+ (nonnull instancetype) nodeWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
```

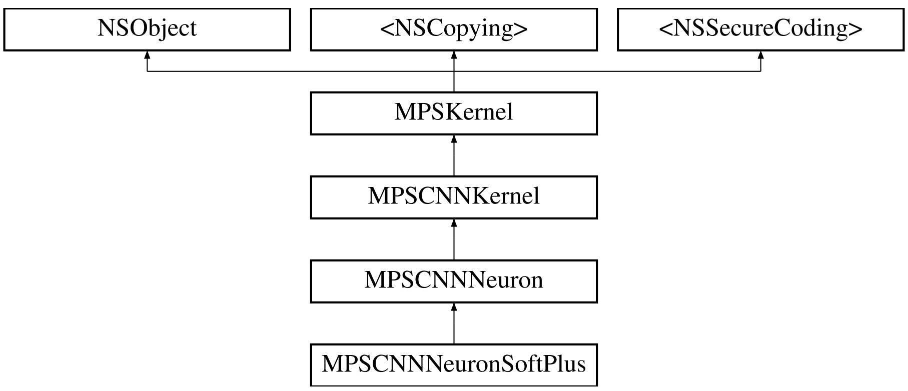
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.41 MPSCNNNeuronSoftPlus Class Reference

```
#import <MPSCNNConvolution.h>
```

Inheritance diagram for MPSCNNNeuronSoftPlus:



Instance Methods

- (nonnull instancetype) - initWithDevice:a:b:
- (nonnull instancetype) - initWithDevice:

Properties

- float a
- float b

Additional Inherited Members

5.41.1 Detailed Description

This depends on Metal.framework Specifies the parametric softplus neuron filter. For each pixel, applies the following function: f(x) = a ∗ log(1 + e^(b ∗ x))

5.41.2 Method Documentation

5.41.2.1 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

A pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSCNNKernel.

5.41.2.2 initWithDevice:a:b:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            a:(float) a
            b:(float) b
```

Initialize a parametric softplus neuron filter

Parameters

device	The device the filter will run on
a	Filter property "a". See class discussion.
b	Filter property "b". See class discussion.

Returns

A valid MPSCNNNeuronSoftPlus object or nil, if failure.

5.41.3 Property Documentation

5.41.3.1 a

```
- (float) a  [read], [atomic], [assign]
```

5.41.3.2 b

```
- (float) b  [read], [atomic], [assign]
```

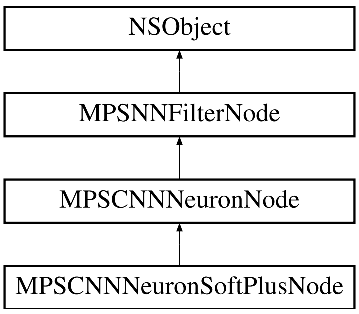
The documentation for this class was generated from the following file:

- MPSCNNConvolution.h

5.42 MPSCNNNeuronSoftPlusNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSCNNNeuronSoftPlusNode:



Instance Methods

- (nonnull instancetype) - initWithSource:a:b:

Class Methods

- (nonnull instancetype) + nodeWithSource:a:b:

Additional Inherited Members

5.42.1 Detailed Description

A node representing a MPSCNNNeuronSoftPlus kernel For each pixel, applies the following function:

```
f(x) = a * log(1 + e^(b * x))
```

5.42.2 Method Documentation

5.42.2.1 initWithSource:a:b:()

```
- (nonnull instancetype) initWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            a:(float) a
            b:(float) b
```

Init a node representing a MPSCNNNeuronSoftPlus kernel

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter
a	See discussion above.
b	See discussion above.

Returns

A new MPSNNFilter node for a MPSCNNNeuronSoftPlus kernel.

5.42.2.2 nodeWithSource:a:b:()

```
+ (nonnull instancetype) nodeWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
```

```
            a:(float) a
            b:(float) b
```

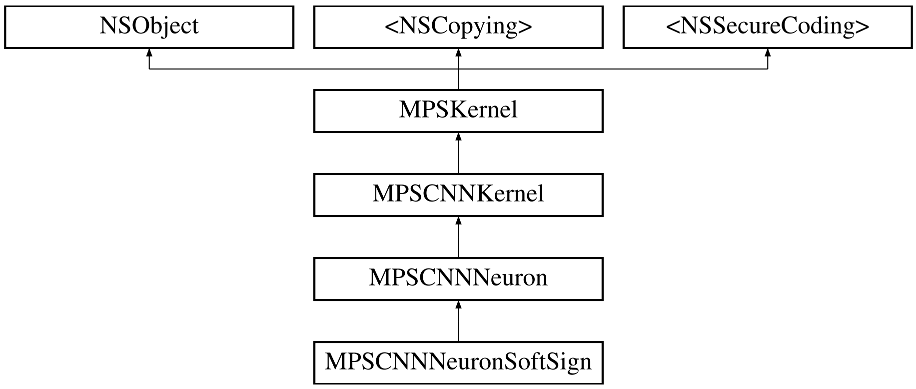
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.43 MPSCNNNeuronSoftSign Class Reference

```
#import <MPSCNNConvolution.h>
```

Inheritance diagram for MPSCNNNeuronSoftSign:



Instance Methods

- (nonnull instancetype) - initWithDevice:

Additional Inherited Members

5.43.1 Detailed Description

This depends on Metal.framework Specifies the softsign neuron filter. For each pixel, applies the following function: f(x) = x / (1 + abs(x))

5.43.2 Method Documentation

5.43.2.1 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Initialize a softsign neuron filter

Parameters

device	The device the filter will run on

Returns

A valid MPSCNNNeuronSoftSign object or nil, if failure.

Reimplemented from MPSCNNKernel.

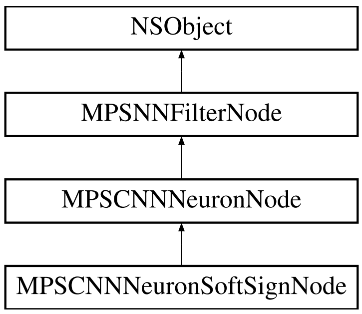
The documentation for this class was generated from the following file:

- MPSCNNConvolution.h

5.44 MPSCNNNeuronSoftSignNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSCNNNeuronSoftSignNode:



Class Methods

- (nonnull instancetype) + nodeWithSource:

Additional Inherited Members

5.44.1 Detailed Description

A node representing a MPSCNNNeuronSoftSign kernel For each pixel, applies the following function:

```
f(x) = x / (1 + abs(x))
```

5.44.2 Method Documentation

5.44.2.1 nodeWithSource:()

```
+ (nonnull instancetype) nodeWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
```

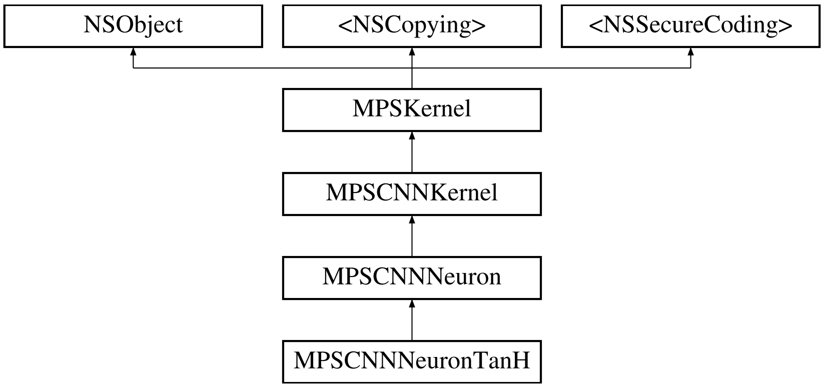
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.45 MPSCNNNeuronTanH Class Reference

```
#import <MPSCNNConvolution.h>
```

Inheritance diagram for MPSCNNNeuronTanH:



Instance Methods

- (nonnull instancetype) - initWithDevice:a:b:
- (nonnull instancetype) - initWithDevice:

Properties

- float a
- float b

Additional Inherited Members

5.45.1 Detailed Description

This depends on Metal.framework Specifies the hyperbolic tangent neuron filter. For each pixel, applies the following function: f(x) = a ∗ tanh(b ∗ x)

5.45.2 Method Documentation

5.45.2.1 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

A pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSCNNKernel.

5.45.2.2 initWithDevice:a:b:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            a:(float) a
            b:(float) b
```

Initialize the hyperbolic tangent neuron filter

Parameters

device	The device the filter will run on
a	Filter property "a". See class discussion.
b	Filter property "b". See class discussion.

Returns

A valid MPSCNNNeuronTanH object or nil, if failure.

5.45.3 Property Documentation

5.45.3.1 a

```
- (float) a  [read], [atomic], [assign]
```

5.45.3.2 b

```
- (float) b  [read], [atomic], [assign]
```

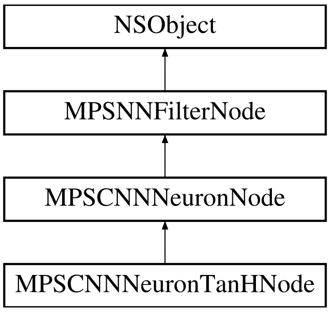
The documentation for this class was generated from the following file:

- MPSCNNConvolution.h

5.46 MPSCNNNeuronTanHNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSCNNNeuronTanHNode:



Instance Methods

- (nonnull instancetype) - initWithSource:a:b:

Class Methods

- (nonnull instancetype) + nodeWithSource:a:b:

Additional Inherited Members

5.46.1 Detailed Description

A node representing a MPSCNNNeuronTanH kernel For each pixel, applies the following function:

```
f(x) = a * tanh(b * x)
```

5.46.2 Method Documentation

5.46.2.1 initWithSource:a:b:()

```
- (nonnull instancetype) initWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            a:(float) a
            b:(float) b
```

Init a node representing a MPSCNNNeuronTanH kernel For each pixel, applies the following function:

```
f(x) = a * tanh(b * x)
```

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter
a	See discussion above.
b	See discussion above.

Returns

A new MPSNNFilter node for a MPSCNNNeuronTanH kernel.

5.46.2.2 nodeWithSource:a:b:()

```
+ (nonnull instancetype) nodeWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            a:(float) a
            b:(float) b
```

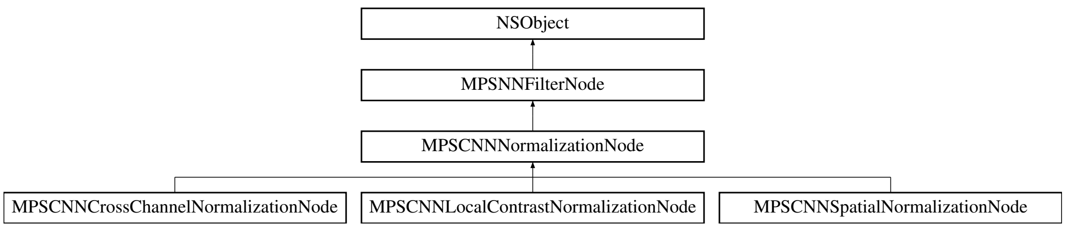
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.47 MPSCNNNormalizationNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSCNNNormalizationNode:



Instance Methods

- (nonnull instancetype) - initWithSource:

Class Methods

- (nonnull instancetype) + nodeWithSource:

Properties

- float alpha
- float beta
- float delta

5.47.1 Detailed Description

virtual base class for CNN normalization nodes

5.47.2 Method Documentation

5.47.2.1 initWithSource:()

```
- (nonnull instancetype) initWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
```

Implemented in MPSCNNCrossChannelNormalizationNode, MPSCNNLocalContrastNormalizationNode, and M↵ PSCNNSpatialNormalizationNode.

5.47.2.2 nodeWithSource:()

```
+ (nonnull instancetype) nodeWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
```

5.47.3 Property Documentation

5.47.3.1 alpha

```
- (float) alpha  [read], [write], [nonatomic], [assign]
```

5.47.3.2 beta

```
- (float) beta  [read], [write], [nonatomic], [assign]
```

5.47.3.3 delta

```
- (float) delta [read], [write], [nonatomic], [assign]
```

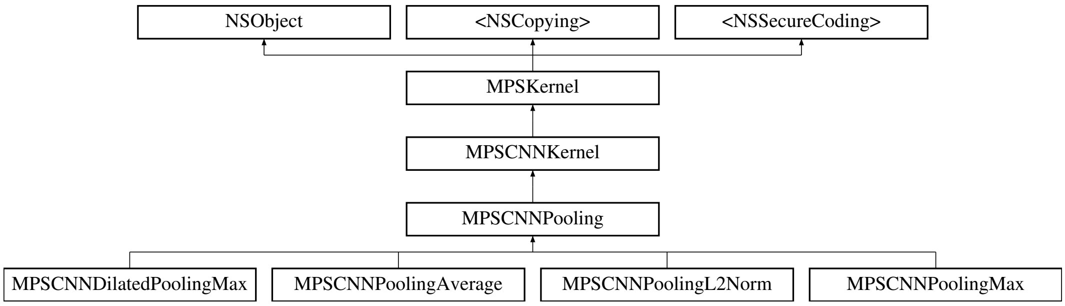
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.48 MPSCNNPooling Class Reference

```
#import <MPSCNNPooling.h>
```

Inheritance diagram for MPSCNNPooling:



Instance Methods

- (nonnull instancetype) - initWithDevice:kernelWidth:kernelHeight:
- (nonnull instancetype) - initWithDevice:kernelWidth:kernelHeight:strideInPixelsX:strideInPixelsY:
- (nullable instancetype) - initWithCoder:device:
- (nonnull instancetype) - initWithDevice:

Additional Inherited Members

5.48.1 Detailed Description

This depends on Metal.framework Pooling is a form of non-linear sub-sampling. Pooling partitions the input image into a set of rectangles (overlapping or non-overlapping) and, for each such sub-region, outputs a value. The pooling operation is used in computer vision to reduce the dimensionality of intermediate representations.

5.48.2 Method Documentation

5.48.2.1 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability See MPSKernel::initWithCoder.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSCNNPooling
device	The MTLDevice on which to make the MPSCNNPooling

Returns

A new MPSCNNPooling object, or nil if failure.

Reimplemented from MPSCNNKernel.

Reimplemented in MPSCNNDilatedPoolingMax, MPSCNNPoolingAverage, MPSCNNPoolingL2Norm, and MPS↩ CNNPoolingMax.

5.48.2.2 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

A pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSCNNKernel.

5.48.2.3 initWithDevice:kernelWidth:kernelHeight:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            kernelWidth:(NSUInteger) kernelWidth
            kernelHeight:(NSUInteger) kernelHeight
```

Initialize a pooling filter

Parameters

device	The device the filter will run on
kernelWidth	The width of the kernel. Can be an odd or even value.
kernelHeight	The height of the kernel. Can be an odd or even value.

Returns

A valid MPSCNNPooling object or nil, if failure.

5.48.2.4 initWithDevice:kernelWidth:kernelHeight:strideInPixelsX:strideInPixelsY:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            kernelWidth:(NSUInteger) kernelWidth
            kernelHeight:(NSUInteger) kernelHeight
            strideInPixelsX:(NSUInteger) strideInPixelsX
            strideInPixelsY:(NSUInteger) strideInPixelsY
```

Initialize a pooling filter

Parameters

device	The device the filter will run on
kernelWidth	The width of the kernel. Can be an odd or even value.
kernelHeight	The height of the kernel. Can be an odd or even value.
strideInPixelsX	The output stride (downsampling factor) in the x dimension.
strideInPixelsY	The output stride (downsampling factor) in the y dimension.

Returns

A valid MPSCNNPooling object or nil, if failure.

Reimplemented in MPSCNNPoolingAverage, MPSCNNPoolingL2Norm, and MPSCNNPoolingMax.

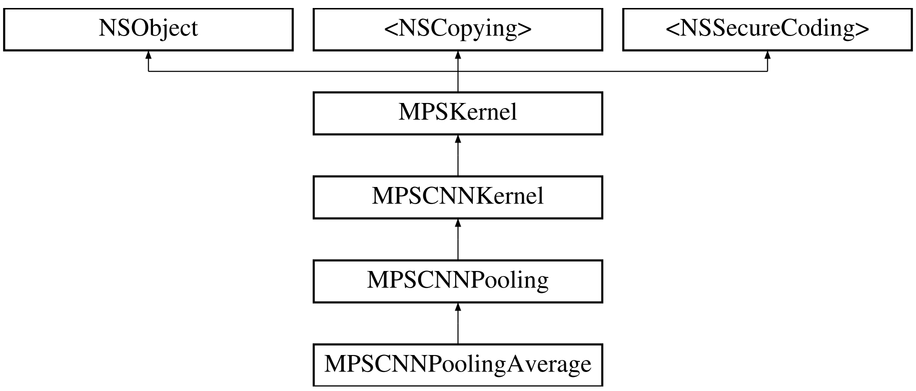
The documentation for this class was generated from the following file:

- MPSCNNPooling.h

5.49 MPSCNNPoolingAverage Class Reference

```
#import <MPSCNNPooling.h>
```

Inheritance diagram for MPSCNNPoolingAverage:



Instance Methods

- (nonnull instancetype) - initWithDevice:kernelWidth:kernelHeight:strideInPixelsX:strideInPixelsY:
- (nullable instancetype) - initWithCoder:device:

Properties

- NSUInteger zeroPadSizeX
- NSUInteger zeroPadSizeY

Additional Inherited Members

5.49.1 Detailed Description

This depends on Metal.framework Specifies the average pooling filter. For each pixel, returns the mean value of pixels in the kernelWidth x kernelHeight filter region. When edgeMode is MPSImageEdgeModeClamp the filtering window is shrunk to remain

within the source image borders. What this means is that close to image borders the filtering window

will be smaller in order to fit inside the source image and less values will be used to compute the average. In case the filtering window is entirely outside the source image border the outputted value will be zero.

5.49.2 Method Documentation

5.49.2.1 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability See MPSKernel::initWithCoder.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSCNNPooling
device	The MTLDevice on which to make the MPSCNNPooling

Returns

A new MPSCNNPooling object, or nil if failure.

Reimplemented from MPSCNNPooling.

5.49.2.2 initWithDevice:kernelWidth:kernelHeight:strideInPixelsX:strideInPixelsY:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            kernelWidth:(NSUInteger) kernelWidth
            kernelHeight:(NSUInteger) kernelHeight
            strideInPixelsX:(NSUInteger) strideInPixelsX
            strideInPixelsY:(NSUInteger) strideInPixelsY
```

Initialize a MPSCNNPoolingAverage pooling filter

Parameters

device	The device the filter will run on
kernelWidth	The width of the kernel. Can be an odd or even value.
kernelHeight	The height of the kernel. Can be an odd or even value.
strideInPixelsX	The output stride (downsampling factor) in the x dimension.
strideInPixelsY	The output stride (downsampling factor) in the y dimension.

Returns

A valid MPSCNNPooling object or nil, if failure.

Reimplemented from MPSCNNPooling.

5.49.3 Property Documentation

5.49.3.1 zeroPadSizeX

```
- zeroPadSizeX [read], [write], [nonatomic], [assign]
```

How much zero padding to apply to both left and right borders of the input image for average pooling, when using

See also

edgeMode MPSImageEdgeModeClamp. For
edgeMode MPSImageEdgeModeZero this property is ignored and the area outside the image is interpreted to contain zeros. The zero padding size is used to shrink the pooling window to fit inside the area bound by the source image and its padding region, but the effect is that the normalization factor of the average computation is computed also for the zeros in the padding region.

5.49.3.2 zeroPadSizeY

```
- zeroPadSizeY  [read], [write], [nonatomic], [assign]
```

How much zero padding to apply to both top and bottom borders of the input image for average pooling, when using

See also

edgeMode MPSImageEdgeModeClamp. For
edgeMode MPSImageEdgeModeZero this property is ignored and the area outside the image is interpreted to contain zeros. The zero padding size is used to shrink the pooling window to fit inside the area bound by the source image and its padding region, but the effect is that the normalization factor of the average computation is computed also for the zeros in the padding region.

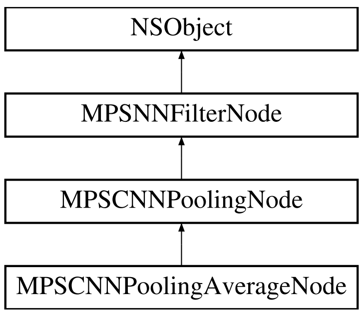
The documentation for this class was generated from the following file:

- MPSCNNPooling.h

5.50 MPSCNNPoolingAverageNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSCNNPoolingAverageNode:



Additional Inherited Members

5.50.1 Detailed Description

A node representing a MPSCNNPoolingAverage kernel

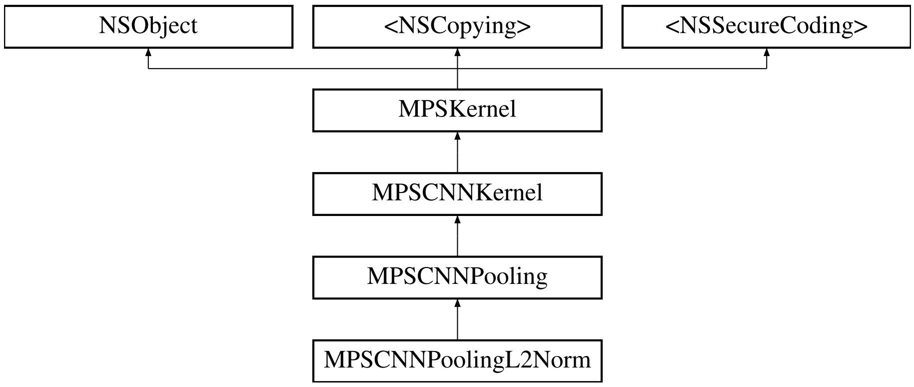
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.51 MPSCNNPoolingL2Norm Class Reference

```
#import <MPSCNNPooling.h>
```

Inheritance diagram for MPSCNNPoolingL2Norm:



Instance Methods

- (nonnull instancetype) - initWithDevice:kernelWidth:kernelHeight:strideInPixelsX:strideInPixelsY:
- (nullable instancetype) - initWithCoder:device:

Additional Inherited Members

5.51.1 Detailed Description

This depends on Metal.framework Specifies the L2-norm pooling filter. For each pixel, returns L2-Norm of pixels in the kernelWidth x kernelHeight filter region. out[c,x,y] = sqrt (sum_{dx,dy} in[c,x+dx,y+dy] ∗ in[c,x+dx,y+dy]).

5.51.2 Method Documentation

5.51.2.1 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability See MPSKernel::initWithCoder.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSCNNPooling
device	The MTLDevice on which to make the MPSCNNPooling

Returns

A new MPSCNNPooling object, or nil if failure.

Reimplemented from MPSCNNPooling.

5.51.2.2 initWithDevice:kernelWidth:kernelHeight:strideInPixelsX:strideInPixelsY:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            kernelWidth:(NSUInteger) kernelWidth
            kernelHeight:(NSUInteger) kernelHeight
            strideInPixelsX:(NSUInteger) strideInPixelsX
            strideInPixelsY:(NSUInteger) strideInPixelsY
```

Initialize a MPSCNNPoolingL2Norm pooling filter

Parameters

device	The device the filter will run on
kernelWidth	The width of the kernel. Can be an odd or even value.
kernelHeight	The height of the kernel. Can be an odd or even value.
strideInPixelsX	The output stride (downsampling factor) in the x dimension.
strideInPixelsY	The output stride (downsampling factor) in the y dimension.

Returns

A valid MPSCNNPooling object or nil, if failure.

Reimplemented from MPSCNNPooling.

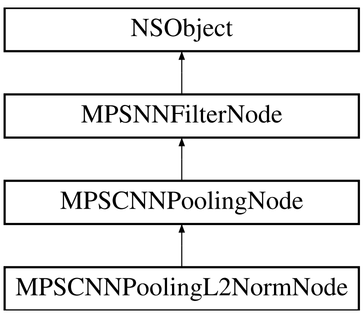
The documentation for this class was generated from the following file:

- MPSCNNPooling.h

5.52 MPSCNNPoolingL2NormNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSCNNPoolingL2NormNode:



Additional Inherited Members

5.52.1 Detailed Description

A node representing a MPSCNNPoolingL2Norm kernel

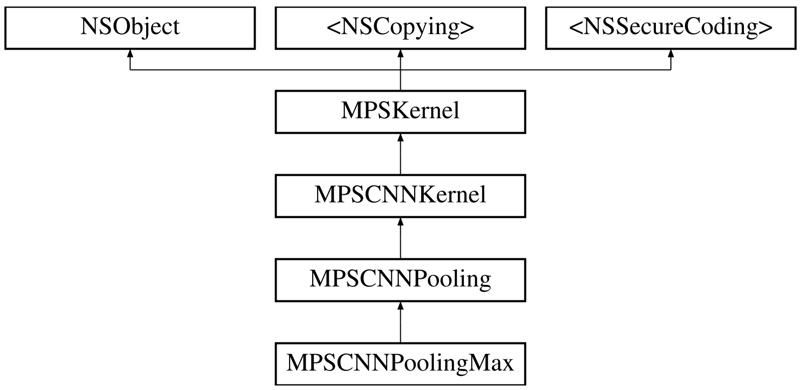
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.53 MPSCNNPoolingMax Class Reference

```
#import <MPSCNNPooling.h>
```

Inheritance diagram for MPSCNNPoolingMax:



Instance Methods

- (nonnull instancetype) - initWithDevice:kernelWidth:kernelHeight:strideInPixelsX:strideInPixelsY:
- (nullable instancetype) - initWithCoder:device:

Additional Inherited Members

5.53.1 Detailed Description

This depends on Metal.framework Specifies the max pooling filter. For each pixel, returns the maximum value of pixels in the kernelWidth x kernelHeight filter region.

5.53.2 Method Documentation

5.53.2.1 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability See MPSKernel::initWithCoder.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSCNNPooling
device	The MTLDevice on which to make the MPSCNNPooling

Returns

A new MPSCNNPooling object, or nil if failure.

Reimplemented from MPSCNNPooling.

5.53.2.2 initWithDevice:kernelWidth:kernelHeight:strideInPixelsX:strideInPixelsY:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            kernelWidth:(NSUInteger) kernelWidth
            kernelHeight:(NSUInteger) kernelHeight
            strideInPixelsX:(NSUInteger) strideInPixelsX
            strideInPixelsY:(NSUInteger) strideInPixelsY
```

Initialize a MPSCNNPoolingMax pooling filter

Parameters

device	The device the filter will run on
kernelWidth	The width of the kernel. Can be an odd or even value.
kernelHeight	The height of the kernel. Can be an odd or even value.
strideInPixelsX	The output stride (downsampling factor) in the x dimension.
strideInPixelsY	The output stride (downsampling factor) in the y dimension.

Returns

A valid MPSCNNPooling object or nil, if failure.

Reimplemented from MPSCNNPooling.

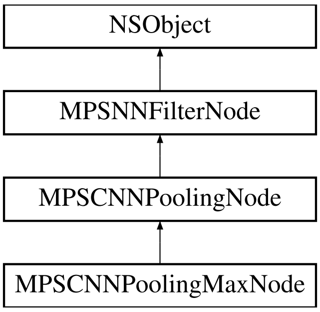
The documentation for this class was generated from the following file:

- MPSCNNPooling.h

5.54 MPSCNNPoolingMaxNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSCNNPoolingMaxNode:



Additional Inherited Members

5.54.1 Detailed Description

A node representing a MPSCNNPoolingMax kernel

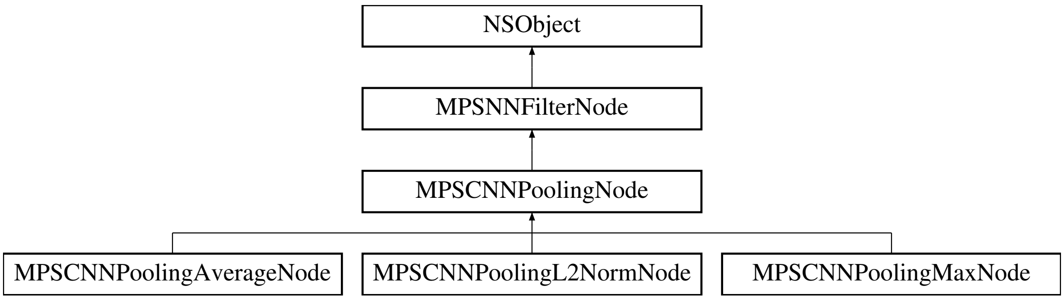
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.55 MPSCNNPoolingNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSCNNPoolingNode:



Instance Methods

- (nonnull instancetype) - initWithSource:kernelWidth:kernelHeight:strideInPixelsX:strideInPixelsY:
- (nonnull instancetype) - initWithSource:filterSize:stride:
- (nonnull instancetype) - initWithSource:filterSize:

Class Methods

- (nonnull instancetype) + nodeWithSource:filterSize:
- (nonnull instancetype) + nodeWithSource:filterSize:stride:

Additional Inherited Members

5.55.1 Detailed Description

A node for a MPSCNNPooling kernel This is an abstract base class that does not correspond with any particular MPSCNNKernel. Please make one of the MPSCNNPooling subclasses instead.

5.55.2 Method Documentation

5.55.2.1 initWithSource:filterSize:()

```
- (nonnull instancetype) initWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            filterSize:(NSUInteger) size
```

Convenience initializer for MPSCNNPooling nodes with square non-overlapping kernels

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter
size	kernelWidth = kernelHeight = strideInPixelsX = strideInPixelsY = size

Returns

A new MPSNNFilter node for a MPSCNNPooling kernel.

5.55.2.2 initWithSource:filterSize:stride:()

```
- (nonnull instancetype) initWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            filterSize:(NSUInteger) size
            stride:(NSUInteger) stride
```

Convenience initializer for MPSCNNPooling nodes with square kernels

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter
size	kernelWidth = kernelHeight = size
stride	strideInPixelsX = strideInPixelsY = stride

Returns

A new MPSNNFilter node for a MPSCNNPooling kernel.

5.55.2.3 initWithSource:kernelWidth:kernelHeight:strideInPixelsX:strideInPixelsY:()

```
- (nonnull instancetype) initWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            kernelWidth:(NSUInteger) kernelWidth
            kernelHeight:(NSUInteger) kernelHeight
            strideInPixelsX:(NSUInteger) strideInPixelsX
            strideInPixelsY:(NSUInteger) strideInPixelsY
```

Init a node representing a MPSCNNPooling kernel

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter
kernelWidth	The width of the max filter window
kernelHeight	The height of the max filter window
strideInPixelsX	The output stride (downsampling factor) in the x dimension.
strideInPixelsY	The output stride (downsampling factor) in the y dimension.

Returns

A new MPSNNFilter node for a MPSCNNPooling kernel.

5.55.2.4 nodeWithSource:filterSize:()

```
+ (nonnull instancetype) nodeWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            filterSize:(NSUInteger) size
```

Convenience initializer for MPSCNNPooling nodes with square non-overlapping kernels

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter
size	kernelWidth = kernelHeight = strideInPixelsX = strideInPixelsY = size

Returns

A new MPSNNFilter node for a MPSCNNPooling kernel.

5.55.2.5 nodeWithSource:filterSize:stride:()

```
+ (nonnull instancetype) nodeWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            filterSize:(NSUInteger) size
            stride:(NSUInteger) stride
```

Convenience initializer for MPSCNNPooling nodes with square non-overlapping kernels and a different stride

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter
size	kernelWidth = kernelHeight = size
stride	strideInPixelsX = strideInPixelsY = stride

Returns

A new MPSNNFilter node for a MPSCNNPooling kernel.

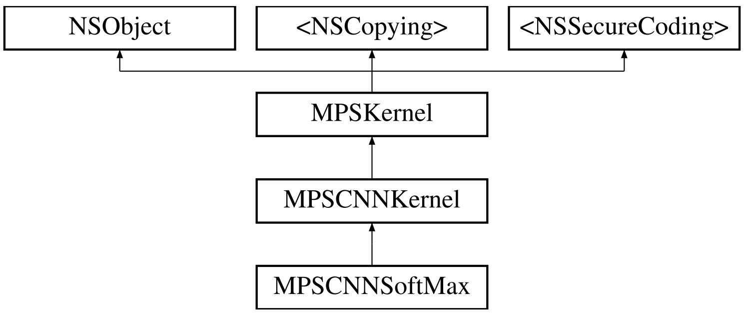
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.56 MPSCNNSoftMax Class Reference

```
#import <MPSCNNSoftMax.h>
```

Inheritance diagram for MPSCNNSoftMax:



Additional Inherited Members

5.56.1 Detailed Description

This depends on Metal.framework The softmax filter is a neural transfer function and is useful for classification tasks. The softmax filter is applied across feature channels and in a convolutional manner at all spatial locations. The softmax filter can be seen as the combination of an activation function (exponential) and a normalization operator. For each feature channel per pixel in an image in a feature map, the softmax filter computes the following: result channel in pixel = exp(pixel(x,y,k))/sum(exp(pixel(x,y,0)) ... exp(pixel(x,y,N-1)) where N is the number of feature channels

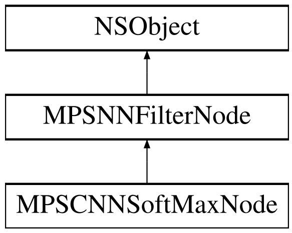
The documentation for this class was generated from the following file:

- MPSCNNSoftMax.h

5.57 MPSCNNSoftMaxNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSCNNSoftMaxNode:



Instance Methods

- (nonnull instancetype) - initWithSource:

Class Methods

- (nonnull instancetype) + nodeWithSource:

Additional Inherited Members

5.57.1 Detailed Description

Node representing a MPSCNNSoftMax kernel

5.57.2 Method Documentation

5.57.2.1 initWithSource:()

```
- (nonnull instancetype) initWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
```

Init a node representing a MPSCNNSoftMax kernel

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter

Returns

A new MPSNNFilter node for a MPSCNNSoftMax kernel.

5.57.2.2 nodeWithSource:()

```
+ (nonnull instancetype) nodeWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
```

Init a node representing a autoreleased MPSCNNSoftMax kernel

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter

Returns

A new MPSNNFilter node for a MPSCNNSoftMax kernel.

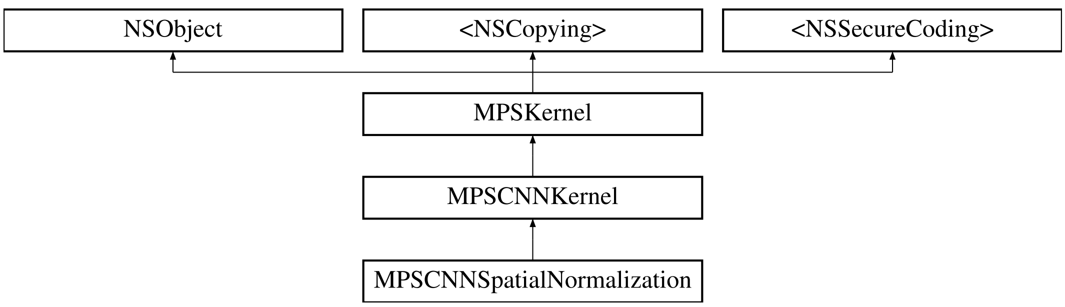
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.58 MPSCNNSpatialNormalization Class Reference

```
#import <MPSCNNNormalization.h>
```

Inheritance diagram for MPSCNNSpatialNormalization:



Instance Methods

- (nonnull instancetype) - initWithDevice:kernelWidth:kernelHeight:
- (nullable instancetype) - initWithCoder:device:
- (nonnull instancetype) - initWithDevice:

Properties

- float alpha
- float beta
- float delta
- NSUInteger kernelWidth
- NSUInteger kernelHeight

Additional Inherited Members

5.58.1 Detailed Description

This depends on Metal.framework Specifies the spatial normalization filter. The spatial normalization for a feature channel applies the filter over local regions which extend spatially, but are in separate feature channels (i.e., they have shape 1 x kernelWidth x kernelHeight). For each feature channel, the function computes the sum of squares of X inside each rectangle, N2(i,j). It then divides each element of X as follows: Y(i,j) = X(i,j) / (delta + alpha/(kw∗kh) ∗ N2(i,j))^beta, where kw and kh are the kernelWidth and the kernelHeight. It is the end-users responsibility to ensure that the combination of the parameters delta and alpha does not result in a situation where the denominator becomes zero - in such situations the resulting pixel-value is undefined.

5.58.2 Method Documentation

5.58.2.1 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSCNNKernel.

5.58.2.2 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

A pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSCNNKernel.

5.58.2.3 initWithDevice:kernelWidth:kernelHeight:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            kernelWidth:(NSUInteger) kernelWidth
            kernelHeight:(NSUInteger) kernelHeight
```

Initialize a spatial normalization filter

Parameters

device	The device the filter will run on
kernelWidth	The width of the kernel
kernelHeight	The height of the kernel

Returns

A valid MPSCNNSpatialNormalization object or nil, if failure.

NOTE: For now, kernelWidth must be equal to kernelHeight

5.58.3 Property Documentation

5.58.3.1 alpha

```
- alpha  [read], [write], [nonatomic], [assign]
```

The value of alpha. Default is 1.0. Must be non-negative.

5.58.3.2 beta

`- beta [read], [write], [nonatomic], [assign]`

The value of beta. Default is 5.0

5.58.3.3 delta

`- delta [read], [write], [nonatomic], [assign]`

The value of delta. Default is 1.0

5.58.3.4 kernelHeight

`- kernelHeight [read], [nonatomic], [assign]`

The height of the filter window

5.58.3.5 kernelWidth

`- kernelWidth [read], [nonatomic], [assign]`

The width of the filter window

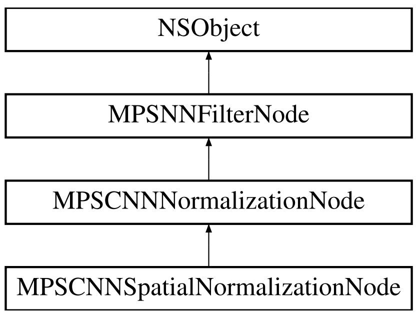
The documentation for this class was generated from the following file:

- MPSCNNNormalization.h

5.59 MPSCNNSpatialNormalizationNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSCNNSpatialNormalizationNode:



Instance Methods

- (nonnull instancetype) - initWithSource:kernelSize:
- (nonnull instancetype) - initWithSource:

Class Methods

- (nonnull instancetype) + nodeWithSource:kernelSize:

Properties

- NSUInteger kernelWidth
- NSUInteger kernelHeight

5.59.1 Method Documentation

5.59.1.1 initWithSource:()

```
- (nonnull instancetype) initWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
```

Implements MPSCNNNormalizationNode.

5.59.1.2 initWithSource:kernelSize:()

```
- (nonnull instancetype) initWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            kernelSize:(NSUInteger) kernelSize
```

5.59.1.3 nodeWithSource:kernelSize:()

```
+ (nonnull instancetype) nodeWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            kernelSize:(NSUInteger) kernelSize
```

5.59.2 Property Documentation

5.59.2.1 kernelHeight

```
- (NSUInteger) kernelHeight  [read], [write], [nonatomic], [assign]
```

5.59.2.2 kernelWidth

```
- (NSUInteger) kernelWidth [read], [write], [nonatomic], [assign]
```

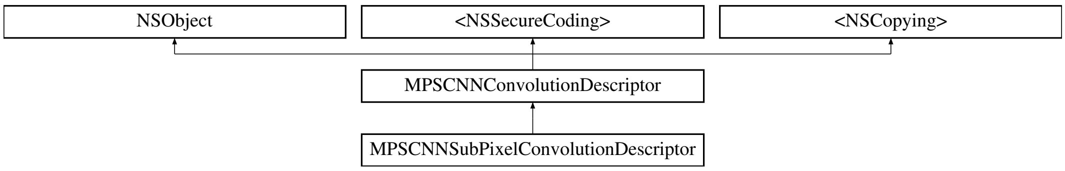
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.60 MPSCNNSubPixelConvolutionDescriptor Class Reference

```
#import <MPSCNNConvolution.h>
```

Inheritance diagram for MPSCNNSubPixelConvolutionDescriptor:



Properties

- NSUInteger subPixelScaleFactor

Additional Inherited Members

5.60.1 Property Documentation

5.60.1.1 subPixelScaleFactor

```
- subPixelScaleFactor [read], [write], [nonatomic], [assign]
```

Upsampling scale factor. Each pixel in input is upsampled into a subPixelScaleFactor x subPixelScaleFactor pixel block by rearranging the outputFeatureChannels as described above. Default value is 1.

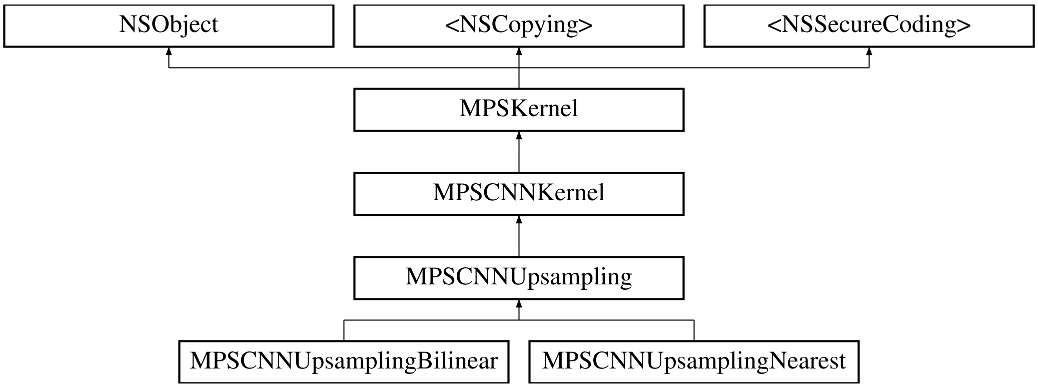
The documentation for this class was generated from the following file:

- MPSCNNConvolution.h

5.61 MPSCNNUpsampling Class Reference

```
#import <MPSCNNUpsampling.h>
```

Inheritance diagram for MPSCNNUpsampling:



Instance Methods

- (nonnull instancetype) - initWithDevice:

Properties

- double scaleFactorX
- double scaleFactorY

Additional Inherited Members

5.61.1 Detailed Description

This depends on Metal.framework The MPSCNNUpsampling filter can be used to resample an existing MPSImage using a different sampling frequency for the x and y dimensions with the purpose of enlarging the size of an image.

The number of output feature channels remains the same as the number of input feature channels.

The scaleFactor must be an integer value >= 1. The default value is 1. If scaleFactor == 1, the filter acts as a copy kernel.

Nearest and bilinear variants are supported.

5.61.2 Method Documentation

5.61.2.1 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

A pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSCNNKernel.

5.61.3 Property Documentation

5.61.3.1 scaleFactorX

```
- scaleFactorX [read], [nonatomic], [assign]
```

The upsampling scale factor for the x dimension. The default value is 1.

5.61.3.2 scaleFactorY

```
- scaleFactorY [read], [nonatomic], [assign]
```

The upsampling scale factor for the y dimension. The default value is 1.

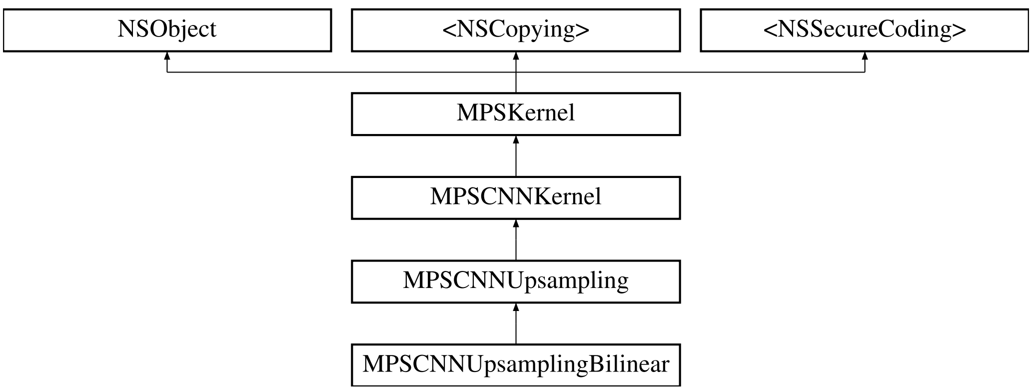
The documentation for this class was generated from the following file:

- MPSCNNUpsampling.h

5.62 MPSCNNUpsamplingBilinear Class Reference

```
#import <MPSCNNUpsampling.h>
```

Inheritance diagram for MPSCNNUpsamplingBilinear:



Instance Methods

- (nonnull instancetype) - initWithDevice:integerScaleFactorX:integerScaleFactorY:

Additional Inherited Members

5.62.1 Detailed Description

This depends on Metal.framework. Specifies the bilinear spatial upsampling filter.

5.62.2 Method Documentation

5.62.2.1 initWithDevice:integerScaleFactorX:integerScaleFactorY:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            integerScaleFactorX:(NSUInteger) integerScaleFactorX
            integerScaleFactorY:(NSUInteger) integerScaleFactorY
```

Initialize the bilinear spatial upsampling filter.

Parameters

device	The device the filter will run on.
integerScaleFactorX	The upsampling factor for the x dimension.
integerScaleFactorY	The upsampling factor for the y dimension.

Returns

A valid MPSCNNUpsamplingBilinear object or nil, if failure.

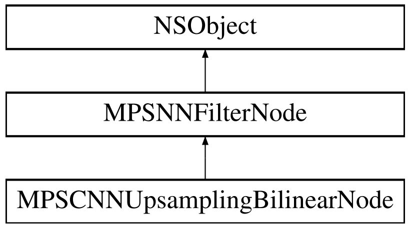
The documentation for this class was generated from the following file:

- MPSCNNUpsampling.h

5.63 MPSCNNUpsamplingBilinearNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSCNNUpsamplingBilinearNode:



Instance Methods

- (nonnull instancetype) - initWithSource:integerScaleFactorX:integerScaleFactorY:

Class Methods

- (nonnull instancetype) + nodeWithSource:integerScaleFactorX:integerScaleFactorY:

Properties

- double scaleFactorX
- double scaleFactorY

5.63.1 Detailed Description

Node representing a MPSCNNUpsamplingBilinear kernel

5.63.2 Method Documentation

5.63.2.1 initWithSource:integerScaleFactorX:integerScaleFactorY:()

```
- (nonnull instancetype) initWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            integerScaleFactorX:(NSUInteger) integerScaleFactorX
            integerScaleFactorY:(NSUInteger) integerScaleFactorY
```

Init a node representing a MPSCNNUpsamplingBilinear kernel

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter
integerScaleFactorX	The upsampling factor for the x dimension.
integerScaleFactorY	The upsampling factor for the y dimension.

Returns

A new MPSNNFilter node for a MPSCNNUpsamplingBilinear kernel.

5.63.2.2 nodeWithSource:integerScaleFactorX:integerScaleFactorY:()

```
+ (nonnull instancetype) nodeWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            integerScaleFactorX:(NSUInteger) integerScaleFactorX
            integerScaleFactorY:(NSUInteger) integerScaleFactorY
```

Init a autoreleased node representing a MPSCNNUpsamplingBilinear kernel

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter
integerScaleFactorX	The upsampling factor for the x dimension.
integerScaleFactorY	The upsampling factor for the y dimension.

Returns

A new MPSNNFilter node for a MPSCNNUpsamplingBilinear kernel.

5.63.3 Property Documentation

5.63.3.1 scaleFactorX

```
- (double) scaleFactorX [read], [nonatomic], [assign]
```

5.63.3.2 scaleFactorY

```
- (double) scaleFactorY [read], [nonatomic], [assign]
```

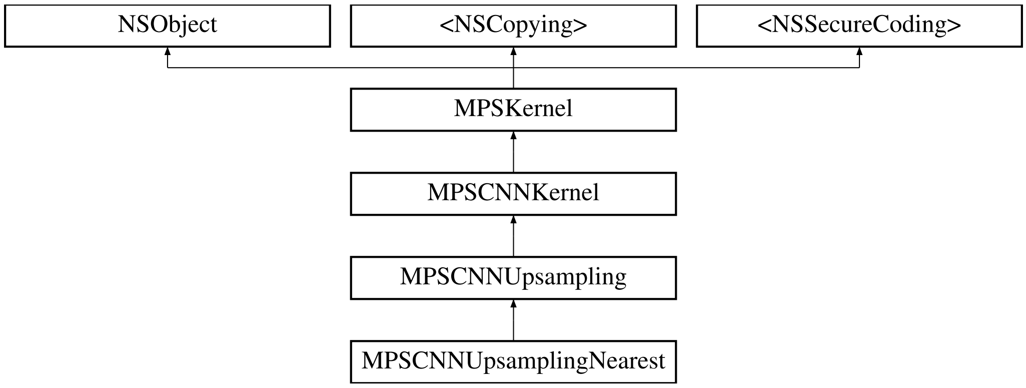
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.64 MPSCNNUpsamplingNearest Class Reference

```
#import <MPSCNNUpsampling.h>
```

Inheritance diagram for MPSCNNUpsamplingNearest:



Instance Methods

- (nonnull instancetype) - initWithDevice:integerScaleFactorX:integerScaleFactorY:

Additional Inherited Members

5.64.1 Detailed Description

This depends on Metal.framework. Specifies the nearest spatial upsampling filter.

5.64.2 Method Documentation

5.64.2.1 initWithDevice:integerScaleFactorX:integerScaleFactorY:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            integerScaleFactorX:(NSUInteger) integerScaleFactorX
            integerScaleFactorY:(NSUInteger) integerScaleFactorY
```

Initialize the nearest spatial upsampling filter.

Parameters

device	The device the filter will run on.
integerScaleFactorX	The upsampling factor for the x dimension.
integerScaleFactorY	The upsampling factor for the y dimension.

Returns

A valid MPSCNNUpsamplingNearest object or nil, if failure.

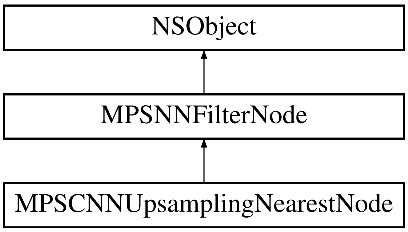
The documentation for this class was generated from the following file:

- MPSCNNUpsampling.h

5.65 MPSCNNUpsamplingNearestNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSCNNUpsamplingNearestNode:



Instance Methods

- (nonnull instancetype) - initWithSource:integerScaleFactorX:integerScaleFactorY:

Class Methods

- (nonnull instancetype) + nodeWithSource:integerScaleFactorX:integerScaleFactorY:

Properties

- double scaleFactorX
- double scaleFactorY

5.65.1 Detailed Description

Node representing a MPSCNNUpsamplingNearest kernel

5.65.2 Method Documentation

5.65.2.1 initWithSource:integerScaleFactorX:integerScaleFactorY:()

```
- (nonnull instancetype) initWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            integerScaleFactorX:(NSUInteger) integerScaleFactorX
            integerScaleFactorY:(NSUInteger) integerScaleFactorY
```

Init a node representing a MPSCNNUpsamplingNearest kernel

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter
integerScaleFactorX	The upsampling factor for the x dimension.
integerScaleFactorY	The upsampling factor for the y dimension.

Returns

A new MPSNNFilter node for a MPSCNNUpsamplingNearest kernel.

5.65.2.2 nodeWithSource:integerScaleFactorX:integerScaleFactorY:()

```
+ (nonnull instancetype) nodeWithSource:
            (MPSNNImageNode *__nonnull) sourceNode
            integerScaleFactorX:(NSUInteger) integerScaleFactorX
            integerScaleFactorY:(NSUInteger) integerScaleFactorY
```

Convenience initializer for an autoreleased MPSCNNUpsamplingNearest nodes

Parameters

sourceNode	The MPSNNImageNode representing the source MPSImage for the filter
integerScaleFactorX	The upsampling factor for the x dimension.
integerScaleFactorY	The upsampling factor for the y dimension.

Returns

A new MPSNNFilter node for a MPSCNNUpsamplingNearest kernel.

5.65.3 Property Documentation

5.65.3.1 scaleFactorX

```
- (double) scaleFactorX [read], [nonatomic], [assign]
```

5.65.3.2 scaleFactorY

```
- (double) scaleFactorY [read], [nonatomic], [assign]
```

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

- MPSNNGraphNodes.h

5.66 <MPSDeviceProvider> Protocol Reference

```
#import <MPSCoreTypes.h>
```

Instance Methods

- (id< MTLDevice >) - mpsMTLDevice

5.66.1 Detailed Description

A way of extending a NSCoder to enable the setting of MTLDevice for unarchived objects When a object is initialized by a NSCoder, it calls -initWithCoder:, which is missing the necessary MTLDevice to correctly initialize the MPSKernel, or MPSNNGraph. If the coder does not conform to MPSDeviceProvider, the system default device will be used. If you would like to specify which device to use, subclass the NSCoder (NSKeyedUnarchiver, etc.) to conform to MPSDeviceProvider so that the device can be gotten from the NSCoder.

5.66.2 Method Documentation

5.66.2.1 mpsMTLDevice()

```
- (id <MTLDevice>) mpsMTLDevice
```

Return the device to use when making MPSKernel subclasses from the NSCoder

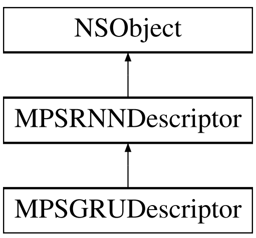
The documentation for this protocol was generated from the following file:

- MPSCoreTypes.h

5.67 MPSGRUDescriptor Class Reference

```
#import <MPSRNNLayer.h>
```

Inheritance diagram for MPSGRUDescriptor:



Class Methods

- (nonnull instancetype) + createGRUDescriptorWithInputFeatureChannels:outputFeatureChannels:

Properties

- id< MPSCNNConvolutionDataSource > inputGateInputWeights
- id< MPSCNNConvolutionDataSource > inputGateRecurrentWeights
- id< MPSCNNConvolutionDataSource > recurrentGateInputWeights
- id< MPSCNNConvolutionDataSource > recurrentGateRecurrentWeights
- id< MPSCNNConvolutionDataSource > outputGateInputWeights
- id< MPSCNNConvolutionDataSource > outputGateRecurrentWeights
- id< MPSCNNConvolutionDataSource > outputGateInputGateWeights
- float gatePnormValue
- BOOL flipOutputGates

5.67.1 Detailed Description

This depends on Metal.framework The MPSGRUDescriptor specifies a GRU (Gated Recurrent Unit) block/layer descriptor. The RNN layer initialized with a MPSGRUDescriptor transforms the input data (image or matrix), and previous output with a set of filters, each producing one feature map in the output data according to the Gated unit formulae detailed below. The user may provide the GRU unit a single input or a sequence of inputs. The layer also supports p-norm gating (Detailed in: https://arxiv.org/abs/1608.03639).

```
Description of operation:
```

Let x_j be the input data (at time index t of sequence, j index containing quadruplet: batch index, x,y and feature index (x=y=0 for matrices)). Let h0_j be the recurrent input (previous output) data from previous time step (at time index t-1 of sequence). Let h_i be the proposed new output. Let h1_i be the output data produced at this time step.

Let Wz_ij, Uz_ij, be the input gate weights for input and recurrent input data respectively Let bi_i be the bias for the input gate

Let Wr_ij, Ur_ij be the recurrent gate weights for input and recurrent input data respectively Let br_i be the bias for the recurrent gate

Let Wh_ij, Uh_ij, Vh_ij, be the output gate weights for input, recurrent gate and input gate respectively Let bh_i be the bias for the output gate

Let gz(x), gr(x), gh(x) be the neuron activation function for the input, recurrent and output gates Let p > 0 be a scalar variable (typicall p >= 1.0) that defines the p-norm gating norm value.

Then the output of the Gated Recurrent Unit layer is computed as follows:

```
    z_i = gz(  Wz_ij * x_j  +  Uz_ij * h0_j  +  bz_i  )
    r_i = gr(  Wr_ij * x_j  +  Ur_ij * h0_j  +  br_i  )
    c_i =      Uh_ij * (r_j h0_j)  +  Vh_ij * (z_j h0_j)
    h_i = gh(  Wh_ij * x_j  + c_i + bh_i   )

h1_i = ( 1 - z_i ^ p)^(1/p) h0_i + z_i h_i
```

The '*' stands for convolution (see MPSRNNImageInferenceLayer) or matrix-vector/matrix multiplication (see MPSRNNMatrixInferenceLayer). Summation is over index j (except for the batch index), but there is no summation over repeated index i - the output index. Note that for validity all intermediate images have to be of same size and all U and V matrices have to be square (ie. outputFeatureChannels == inputFeatureChannels in those). Also the bias terms are scalars wrt. spatial dimensions. The conventional GRU block is achieved by setting Vh = 0 (nil) and the so-called Minimal Gated Unit is achieved with Uh = 0. (The Minimal Gated Unit is detailed in: https://arxiv.org/abs/1603.09420 and there they call z_i the value of the forget gate).

5.67.2 Method Documentation

5.67.2.1 createGRUDescriptorWithInputFeatureChannels:outputFeatureChannels:()

```
+ (nonnull instancetype) createGRUDescriptorWithInputFeatureChannels:
            (NSUInteger) inputFeatureChannels
            outputFeatureChannels:(NSUInteger) outputFeatureChannels
```

Creates a GRU descriptor.

Parameters

inputFeatureChannels	The number of feature channels in the input image/matrix. Must be >= 1.
outputFeatureChannels	The number of feature channels in the output image/matrix. Must be >= 1.

Returns

A valid MPSGRUDescriptor object or nil, if failure.

5.67.3 Property Documentation

5.67.3.1 flipOutputGates

```
- flipOutputGates  [read], [write], [nonatomic], [assign]
```

If YES then the GRU-block output formula is changed to: h1_i = (1 - z_i ^ p)^(1/p) h_i + z_i h0_i. Defaults to NO.

5.67.3.2 gatePnormValue

```
- gatePnormValue  [read], [write], [nonatomic], [assign]
```

The p-norm gating norm value as specified by the GRU formulae. Defaults to 1.0f.

5.67.3.3 inputGateInputWeights

```
- inputGateInputWeights  [read], [write], [nonatomic], [retain]
```

Contains weights 'Wz_ij', bias 'bz_i' and neuron 'gz' from the GRU formula. If nil then assumed zero weights, bias and no neuron (identity mapping). Defaults to nil.

5.67.3.4 inputGateRecurrentWeights

```
- inputGateRecurrentWeights [read], [write], [nonatomic], [retain]
```

Contains weights 'Uz_ij' from the GRU formula. If nil then assumed zero weights. Defaults to nil.

5.67.3.5 outputGateInputGateWeights

```
- outputGateInputGateWeights [read], [write], [nonatomic], [retain]
```

Contains weights 'Vh_ij' - can be used to implement the "Minimally Gated Unit". If nil then assumed zero weights. Defaults to nil.

5.67.3.6 outputGateInputWeights

```
- outputGateInputWeights [read], [write], [nonatomic], [retain]
```

Contains weights 'Wh_ij', bias 'bh_i' and neuron 'gh' from the GRU formula. If nil then assumed zero weights, bias and no neuron (identity mapping).Defaults to nil.

5.67.3.7 outputGateRecurrentWeights

```
- outputGateRecurrentWeights [read], [write], [nonatomic], [retain]
```

Contains weights 'Uh_ij' from the GRU formula. If nil then assumed zero weights. Defaults to nil.

5.67.3.8 recurrentGateInputWeights

```
- recurrentGateInputWeights [read], [write], [nonatomic], [retain]
```

Contains weights 'Wr_ij', bias 'br_i' and neuron 'gr' from the GRU formula. If nil then assumed zero weights, bias and no neuron (identity mapping).Defaults to nil.

5.67.3.9 recurrentGateRecurrentWeights

```
- recurrentGateRecurrentWeights [read], [write], [nonatomic], [retain]
```

Contains weights 'Ur_ij' from the GRU formula. If nil then assumed zero weights.Defaults to nil.

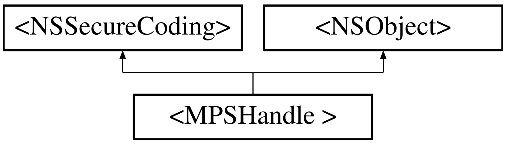
The documentation for this class was generated from the following file:

- MPSRNNLayer.h

5.68 <MPSHandle > Protocol Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for <MPSHandle >:



Instance Methods

- (NSString ∗__nullable) - label

5.68.1 Method Documentation

5.68.1.1 label()

```
- (NSString * __nullable MPSHandle) label    [required]
```

A label to be attached to associated MTLResources for this node

Returns

A human readable string for debugging purposes

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

- MPSNNGraphNodes.h

5.69 <MPSHandle> Protocol Reference

```
#include <MPSNNGraphNodes.h>
```

5.69.1 Detailed Description

MPSNNGraphNodes.h MetalPerformanceShaders

Created by Ian Ollmann on 10/19/16.

Copyright

This header describes building blocks to prepare a graph of MPS images, kernels and state objects. You should prepare your graph by creating a MPSNNImageNode for each of the graph input textures. These are then used as inputs to MPSNNFilterNode subclasses. These in turn produce more image nodes as results, which can be linked to more MPSNNFilterNodes as inputs. When the graph representation is complete, make a MPSNNGraph object to interpret and optimize the graph. The MPSNNGraph may be used to encode the entire graph on a MTLCommand←Buffer.

Objects presented here are generally light weight. They do not have a MTLDevice reference, and so can not create MTLResource objects. In the few cases when data is expected to be large (e.g. convolution weights), the nodes are designed to defer allocation of storage, preferring to leave them on disk or network or other persistent storage to hold the data until it is actually needed to initialize a MPSKernel object. Not until the MPSNNGraph is constructed does the heavy lifting begin. MPSNNGraphs in contrast can be extremely heavy. A large graph may use most of the memory available to your system! Nearly all of this is due to convolution weights. Construct your <MPSCNNConvolutionDataSource> to only load data when it is needed.

MPS resource identification Most of the time, there is only one image and one or fewer states needed as input to a graph, so the order of the images and states passed to [MPSNNGraph encodeToCommandBuffer:sourceImages:] or [MPSNNGraph encodeToCommandBuffer:sourceImages:sourceStates:intermediateImages:destinationStates←:] is clear. There is only one order. However, sometimes graphs have more than one input image or state. What order should they appear in the NSArray passed to these methods?

Each MPSNNImageNode or MPSNNStateNode can be tagged with a MPSHandle. MPSNNGraph keeps track of these. You can request that the MPSNNGraph return them to you in an array in the same order as needed to encode the MPSNNGraph, using MPSNNGraph.sourceImageHandles and MPSNNGraph.sourceStateHandles.

Example:

```
@interface MyHandle : NSObject <MPSHandle>
    // Add a method for my use to get the image needed based on the handle to it.
    // This isn't part of the MPSHandle protocol, but we need it for MyEncodeGraph
    // below. Since it is my code calling my object, we can add whatever we want like this.
    -(MPSImage*__nonnull) image;    // return the MPSImage corresponding to the handle

    // required by MPSHandle protocol
    -(NSString * __nullable) label;

    // MPSHandle implies NSSecureCoding too
    +(BOOL) supportsSecureCoding;
    - (void)encodeWithCoder:(NSCoder * __nonnull )aCoder;
    - (nullable instancetype)initWithCoder:(NSCoder * __nonnull )aDecoder; // NS_DESIGNATED_INITIALIZER
@end

// Encode a graph to cmdBuf using handles for images
// Here we assume that the MPSNNImageNodes that are graph inputs (not produced
// by the graph) are tagged with a unique instance of MyHandle that can be used
// to get the appropriate image for that node.
static void MyEncodeGraph( MPSNNGraph * graph, id <MTLCommandBuffer> cmdBuf )
{
    @autoreleasepool{
        // prepare an array of source images, using the handles
        NSArray<MyHandle*> * handles = graph.sourceImageHandles;
        unsigned long count = handles.count;
        NSMutableArray<MPSImage*> * __nonnull images = [NSMutableArray arrayWithCapacity: count];
        for( unsigned long i = 0; i < count; i++ )
            images[i] = handles[i].image;

        // encode the graph using the array
        [ graph encodeToCommandBuffer: cmdBuf
                        sourceImages: images ];
    }
}
```

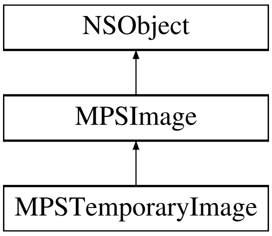
The documentation for this protocol was generated from the following file:

- MPSNNGraphNodes.h

5.70 MPSImage Class Reference

```
#import <MPSImage.h>
```

Inheritance diagram for MPSImage:



Instance Methods

- (nonnull instancetype) - initWithDevice:imageDescriptor:
- (nonnull instancetype) - initWithTexture:featureChannels:
- (nonnull instancetype) - init
- (MPSPurgeableState) - setPurgeableState:
- (void) - readBytes:dataLayout:bytesPerRow:region:featureChannelInfo:imageIndex:
- (void) - writeBytes:dataLayout:bytesPerRow:region:featureChannelInfo:imageIndex:
- (void) - readBytes:dataLayout:imageIndex:
- (void) - writeBytes:dataLayout:imageIndex:

Class Methods

- (nonnull id< MPSImageAllocator >) + defaultAllocator

Properties

- id< MTLDevice > device
- NSUInteger width
- NSUInteger height
- NSUInteger featureChannels
- NSUInteger numberOfImages
- MTLTextureType textureType
- MTLPixelFormat pixelFormat
- NSUInteger precision
- MTLTextureUsage usage
- size_t pixelSize
- id< MTLTexture > texture
- NSString * label

5.70.1 Detailed Description

This depends on Metal.framework A MPSImage object describes a MTLTexture that may have more than 4 channels. Some image types, such as those found in convolutional neural networks (CNN) differ from a standard texture in that they may have more than 4 channels per image. While the channels could hold RGBA data, they will more commonly hold a number of structural permutations upon a multi-channel image as the neural network progresses. It is not uncommon for each pixel to have 32 or 64 channels in it.

A standard MTLTexture may have no more than 4 channels. The additional channels are stored in slices of 2d texture array (i.e. texture type is MTLTextureType2DArray) such that 4 consecutive channels are stored in each slice of this array. If the number of feature channels is N, number of array slices needed is (N+3)/4. E.g. a CNN image with width 3 and height 2 with 9 channels will be stored as

```
slice 0    RGBA   RGBA  RGBA
           RGBA   RGBA  RGBA

slice 1       RGBA   RGBA   RGBA
              RGBA   RGBA   RGBA         (ASCII art /diagonal offset/ intended to show a Z dimension)

slice 2          R???   R???   R???
                 R???   R???   R???
```

The width and height of underlying 2d texture array is the same as the width and height of the MPSImage. The array length is equal to (featureChannels + 3) / 4. Channels marked with ? are just for padding and should not contain NaNs or Infs.

A MPSImage can be container of multiple CNN images for batch processing. In order to create a MPSImage that contains N images, create MPSImageDescriptor with numberOfImages set to N.

Although a MPSImage can contain numberOfImages > 1, the actual number of images among these processed by MPSCNNKernel is controlled by z-dimension of the clipRect. A MPSCNNKernel processes n=clipRect.size.depth images from this collection. The starting source image index to process is given by offset.z. The starting index of the destination image is given by clipRect.origin.z. The MPSCNNKernel takes n=clipRect.size.depth images from tje source at indices [offset.z, offset.z+n], processes each independently and stores the result in the destination at indices [clipRect.origin.z, clipRect.origin.z+n] respectively. Offset.z+n should be <= [src numberOfImage] and clipRect.origin.z+n should be <= [dest numberOfImages] and offset.z must be >= 0.

Example: Suppose MPSCNNConvolution takes an input image with 8 channels and outputs an image with 16 channels. The number of slices needed in the source 2d texture array is 2 and the number of slices needed in the destination 2d array is 4. Suppose the source batch size is 5 and destination batch size is 4. (Multiple N-channel images can be processed concurrently in a batch.) The number of source slices will be 2∗5=10 and number of destination slices will be 4∗4=16. If you want to process just images 2 and 3 of the source and store the result at index 1 and 2 in the destination, you may achieve this by setting offset.z=2, clipRect.origin.z=1 and clipRect.↵ size.depth=2. MPSCNNConvolution will take, in this case, slice 4 and 5 of source and produce slices 4 to 7 of destination. Similarly, slices 6 and 7 will be used to produce slices 8 to 11 of destination.

All MPSCNNKernels process images within each batch independently. That is, calling a MPSCNNKernel on an batch is formally the same as calling it on each image in the batch one at a time. However, quite a lot of CPU and GPU overhead will be avoided if batch processing is used. This is especially important for better performance on small images.

If the number of feature channels is <= 4 and numberOfImages = 1 i.e. only one slice is needed to represent a MPSImage, the underlying metal texture type will be MTLTextureType2D rather than MTLTextureType2DArray.

There are also MPSTemporaryImages, intended for use for very short-lived image data that are produced and consumed immediately in the same MTLCommandBuffer. They are a useful way to minimize CPU-side texture allocation costs and greatly reduce the amount of memory used by your image pipeline.

Creation of the underlying texture may in some cases occur lazily. You should in general avoid calling MPSImage.↵ texture except when unavoidable to avoid materializing memory for longer than necessary. When possible, use the other MPSImage properties to get information about the MPSImage instead.

Most MPSImages of 4 or fewer feature channels can generate quicklooks output in Xcode for easy visualization of image data in the object. MPSTemporaryImages can not.

5.70.2 Method Documentation

5.70.2.1 defaultAllocator()

```
+ (nonnull id <MPSImageAllocator>) defaultAllocator
```

Get a well known MPSImageAllocator that makes MPSImages

Reimplemented in MPSTemporaryImage.

5.70.2.2 init()

```
- (nonnull instancetype) init
```

5.70.2.3 initWithDevice:imageDescriptor:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            imageDescriptor:(const MPSImageDescriptor *__nonnull) imageDescriptor
```

Initialize an empty image object

Parameters

device	The device that the image will be used. May not be NULL.
imageDescriptor	The MPSImageDescriptor. May not be NULL.

Returns

A valid MPSImage object or nil, if failure. Storage to store data needed is allocated lazily on first use of MPSImage or when application calls MPSImage.texture

Reimplemented in MPSTemporaryImage.

5.70.2.4 initWithTexture:featureChannels:()

```
- (nonnull instancetype) initWithTexture:
            (nonnull id< MTLTexture >) texture
            featureChannels:(NSUInteger) featureChannels
```

Initialize an MPSImage object using Metal texture. Metal texture has been created by user for specific number of feature channels and number of images.

Parameters

texture	The MTLTexture allocated by the user to be used as backing for MPSImage.
featureChannels	Number of feature channels this texture contains.

Returns

A valid MPSImage object or nil, if failure. Application can let MPS framework allocate texture with properties specified in imageDescriptor using initWithDevice:MPSImageDescriptor API above. However in memory intensive application, you can save memory (and allocation/deallocation time) by using MPSTemporaryImage where MPS framework aggressively reuse memory underlying textures on same command buffer. See MPSTemporaryImage class for details below. However, in certain cases, application developer may want more control on allocation, placement, reusing/recycling of memory backing textures used in application using Metal Heaps API. In this case, application can create MPSImage from pre-allocated texture using initWithTexture:featureChannels.

MTLTextureType of texture can be MTLTextureType2D ONLY if featureChannels <= 4 in which case MPSImage.numberOfImages is set to 1. Else it should be MTLTextureType2DArray with arrayLength == numberOfImage ∗ ((featureChannels + 3)/4). MPSImage.numberOfImages is set to texture.arrayLength / ((featureChannels + 3)/4).

For MTLTextures containing typical image data which application may obtain from MetalKit or other libraries such as that drawn from a JPEG or PNG, featureChannels should be set to number of valid color channel e.g. for RGB data, even thought MTLPixelFormat will be MTLPixelFormatRGBA, featureChannels should be set to 3.

Reimplemented in MPSTemporaryImage.

5.70.2.5 readBytes:dataLayout:bytesPerRow:region:featureChannelInfo:imageIndex:()

```
- (void) readBytes:
            (void *__nonnull) dataBytes
            dataLayout:(MPSDataLayout) dataLayout
            bytesPerRow:(NSUInteger) bytesPerRow
            region:(MTLRegion) region
            featureChannelInfo:(MPSImageReadWriteParams) featureChannelInfo
            imageIndex:(NSUInteger) imageIndex
```

readBytes Get the values inside MPSImage and put them in the Buffer passed in.

Parameters

dataBytes	The array allocated by the user to be used to put data from MPSImage, the length should be imageWidth ∗ imageHeight ∗ numberOfFeatureChannels and dataType should be inferred from pixelFormat defined in the Image Descriptor.
dataLayout	The enum tells how to layout MPS data in the buffer.
bytesPerRow	Bytes to stride to point to next row(pixel just below current one) in the user buffer.
featureChannelInfo	information user fills in to write to a set of feature channels in the image
imageIndex	Image index in MPSImage to write to.
region	region of the MPSImage to read from. A region is a structure with the origin in the Image from which to start reading values and a size which represents the width and height of the rectangular region to read from. The z direction denotes the number of images, thus for 1 image, origin.z = 0 and size.depth = 1 Use the enum to set data is coming in with what order. The data type will be determined by the pixelFormat defined in the Image Descriptor.

5.70.2.6 readBytes:dataLayout:imageIndex:()

```
- (void) readBytes:
            (void *__nonnull) dataBytes
            dataLayout:(MPSDataLayout) dataLayout
            imageIndex:(NSUInteger) imageIndex
```

readBytes Get the values inside MPSImage and put them in the Buffer passed in.

Parameters

dataBytes	The array allocated by the user to be used to put data from MPSImage, the length should be imageWidth $*$ imageHeight $*$ numberOfFeatureChannels and dataType should be inferred from pixelFormat defined in the Image Descriptor.
dataLayout	The enum tells how to layout MPS data in the buffer.
imageIndex	Image index in MPSImage to read from. Use the enum to set data is coming in with what order. The data type will be determined by the pixelFormat defined in the Image Descriptor. Region is full image, buffer width and height is same as MPSImage width and height.

5.70.2.7 setPurgeableState:()

```
- (MPSPurgeableState) setPurgeableState:
            (MPSPurgeableState) state
```

setPurgeableState Set (or query) the purgeability state of a MPSImage Usage is per [MTLResource setPurgeable↩State:], except that the MTLTexture might be MPSPurgeableStateAllocationDeferred, which means there is no texture to mark volatile / nonvolatile. Attempts to set purgeability on MTLTextures that have not been allocated will be ignored.

5.70.2.8 writeBytes:dataLayout:bytesPerRow:region:featureChannelInfo:imageIndex:()

```
- (void) writeBytes:
            (const void *__nonnull) dataBytes
            dataLayout:(MPSDataLayout) dataLayout
            bytesPerRow:(NSUInteger) bytesPerRow
            region:(MTLRegion) region
            featureChannelInfo:(MPSImageReadWriteParams) featureChannelInfo
            imageIndex:(NSUInteger) imageIndex
```

writeBytes Set the values inside MPSImage with the Buffer passed in.

Parameters

dataBytes	The array allocated by the user to be used to put data from MPSImage, the length should be imageWidth $*$ imageHeight $*$ numberOfFeatureChannels and dataType should be inferred from pixelFormat defined in the Image Descriptor.
dataLayout	The enum tells how to layout MPS data in the buffer.

Parameters

bytesPerRow	Bytes to stride to point to next row(pixel just below current one) in the user buffer.
region	region of the MPSImage to write to. A region is a structure with the origin in the Image from which to start writing values and a size which represents the width and height of the rectangular region to write in. The z direction denotes the number of images, thus for 1 image, origin.z = 0 and size.depth = 1
featureChannelInfo	information user fills in to read from a set of feature channels in the image
imageIndex	Image index in MPSImage to write to. Use the enum to set data is coming in with what order. The data type will be determined by the pixelFormat defined in the Image Descriptor.

5.70.2.9 writeBytes:dataLayout:imageIndex:()

```
- (void) writeBytes:
            (const void *__nonnull) dataBytes
            dataLayout:(MPSDataLayout) dataLayout
            imageIndex:(NSUInteger) imageIndex
```

writeBytes Set the values inside MPSImage with the Buffer passed in.

Parameters

dataBytes	The array allocated by the user to be used to put data from MPSImage, the length should be imageWidth ∗ imageHeight ∗ numberOfFeatureChannels and dataType should be inferred from pixelFormat defined in the Image Descriptor.
dataLayout	The enum tells how to layout MPS data in the buffer.
imageIndex	Image index in MPSImage to write to. Use the enum to set data is coming in with what order. The data type will be determined by the pixelFormat defined in the Image Descriptor. Region is full image, buffer width and height is same as MPSImage width and height.

5.70.3 Property Documentation

5.70.3.1 device

```
- device  [read], [nonatomic], [retain]
```

The device on which the MPSImage will be used

5.70.3.2 featureChannels

```
- featureChannels  [read], [nonatomic], [assign]
```

The number of feature channels per pixel.

5.70.3.3 height

```
- height [read], [nonatomic], [assign]
```

The formal height of the image in pixels.

5.70.3.4 label

```
- label [read], [write], [atomic], [copy]
```

A string to help identify this object.

5.70.3.5 numberOfImages

```
- numberOfImages [read], [nonatomic], [assign]
```

numberOfImages for batch processing

5.70.3.6 pixelFormat

```
- pixelFormat [read], [nonatomic], [assign]
```

The MTLPixelFormat of the underlying texture

5.70.3.7 pixelSize

```
- pixelSize [read], [nonatomic], [assign]
```

Number of bytes from the first byte of one pixel to the first byte of the next pixel in storage order. (Includes padding.)

5.70.3.8 precision

```
- precision [read], [nonatomic], [assign]
```

The number of bits of numeric precision available for each feature channel. This is precision, not size. That is, float is 24 bits, not 32. half precision floating-point is 11 bits, not 16. SNorm formats have one less bit of precision for the sign bit, etc. For formats like MTLPixelFormatB5G6R5Unorm it is the precision of the most precise channel, in this case 6. When this information is unavailable, typically compressed formats, 0 will be returned.

5.70.3.9 texture

```
- texture [read], [nonatomic], [assign]
```

The associated MTLTexture object. This is a 2D texture if numberOfImages is 1 and number of feature channels <= 4. It is a 2D texture array otherwise. To avoid the high cost of premature allocation of the underlying texture, avoid calling this property except when strictly necessary. [MPSCNNKernel encode...] calls typically cause their arguments to become allocated. Likewise, MPSImages initialized with -initWithTexture: featureChannels: have already been allocated.

5.70.3.10 textureType

```
- textureType [read], [nonatomic], [assign]
```

The type of the underlying texture, typically MTLTextureType2D or MTLTextureType2DArray

5.70.3.11 usage

```
- usage [read], [nonatomic], [assign]
```

Description of texture usage.

5.70.3.12 width

```
- width [read], [nonatomic], [assign]
```

The formal width of the image in pixels.

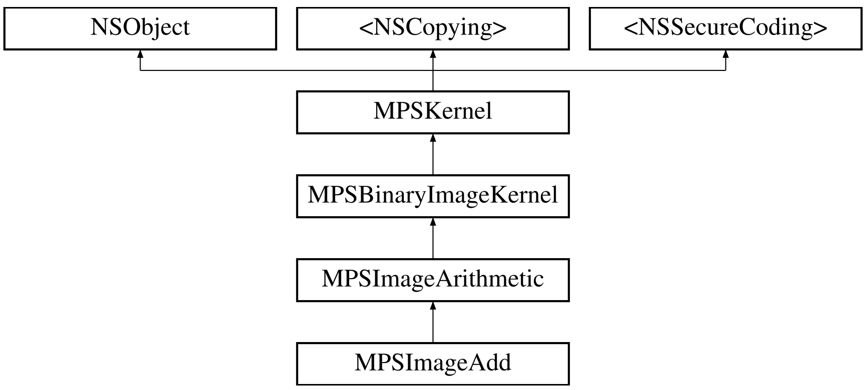
The documentation for this class was generated from the following file:

- MPSCore.framework/Headers/MPSImage.h

5.71 MPSImageAdd Class Reference

```
#import <MPSImageMath.h>
```

Inheritance diagram for MPSImageAdd:



Instance Methods

- (nonnull instancetype) - initWithDevice:

Additional Inherited Members

5.71.1 Detailed Description

This depends on Metal.framework. Specifies the addition operator. For each pixel in the primary source image (x) and each pixel in a secondary source image (y), it applies the following function: result = ((primaryScale $*$ x) + (secondaryScale $*$ y)) + bias.

5.71.2 Method Documentation

5.71.2.1 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Initialize the addition operator

Parameters

device	The device the filter will run on.

Returns

A valid MPSImageAdd object or nil, if failure.

Reimplemented from MPSImageArithmetic.

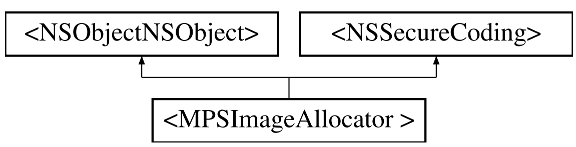
The documentation for this class was generated from the following file:

- MPSImageMath.h

5.72 <MPSImageAllocator > Protocol Reference

```
#import <MPSImage.h>
```

Inheritance diagram for <MPSImageAllocator >:



Instance Methods

- (MPSImage ∗__nonnull) - imageForCommandBuffer:imageDescriptor:kernel:

5.72.1 Detailed Description

A class that allocates new MPSImage or MPSTemporaryImage Sometimes it is prohibitively costly for MPS to figure out how big an image should be in advance. In addition, you may want to have some say over whether the image is a temporary image or not. In such circumstances, the MPSImageAllocator is used to provide the developer with an opportunity for just in time feedback about how the image should be allocated.

Two standard MPSImageAllocators are provided: MPSImageDefaultAllocator and MPSTemporaryImageDefault↩ Allocator. You may of course provide your own allocator instead.

Example:

```
// Note: MPSImageDefaultAllocator is already provided
//       by the framework under that name.  It is provided here
//       as sample code for writing your own variant.
-(MPSImage * __nonnull)  imageForCommandBuffer: (__nonnull id <MTLCommandBuffer>) cmdBuf
                              imageDescriptor: (MPSImageDescriptor * __nonnull)
      descriptor
                                       kernel: (MPSKernel * __nonnull) kernel
{
    MPSImage * result = [[MPSImage alloc] initWithDevice: cmdBuf.device
                                         imageDescriptor: descriptor ];

    // make sure the object sticks around at least as lomg as the command buffer
    [result retain];
    [cmdBuf addCompletedHandler: ^(id <MTLCommandBuffer> c){[result release];}];

    // return autoreleased result
    return [result autorelease];
};

-(BOOL) supportsSecureCoding{ return YES; }
-(void)encodeWithCoder:(NSCoder * __nonnull)aCoder
{
    [super encodeWithCoder: aCoder];

    // encode any data owned by the class at this level
}

-(nullable instancetype) initWithCoder: (NSCoder*__nonnull) aDecoder
{
    self =  [super initWithCoder: aDecoder ];
    if( nil == self )
        return self;

    // use coder to load any extra data kept by this object here

    return self;
}
```

Please see [MPSImage defaultAllocator] and [MPSTemporaryImage defaultAllocator] for implentations of the protocol already provided by MPS.

5.72.2 Method Documentation

5.72.2.1 imageForCommandBuffer:imageDescriptor:kernel:()

```
- (MPSImage ∗ __nonnull MPSImageAllocator) imageForCommandBuffer:
              (__nonnull id< MTLCommandBuffer >) cmdBuf
              imageDescriptor:(MPSImageDescriptor ∗__nonnull) descriptor
              kernel:(MPSKernel ∗__nonnull) kernel   [required]
```

Create a new MPSImage See class description for sample implementation

Parameters

cmdBuf	The MTLCommandBuffer on which the image will be initialized. cmdBuf.device encodes the MTLDevice.
descriptor	A MPSImageDescriptor containing the image format to use. This format is the result of your MPSPadding policy.
kernel	The kernel that will overwrite the image returned by the filter.

Returns

A valid MPSImage or MPSTemporaryImage. It will be automatically released when the command buffer completes.

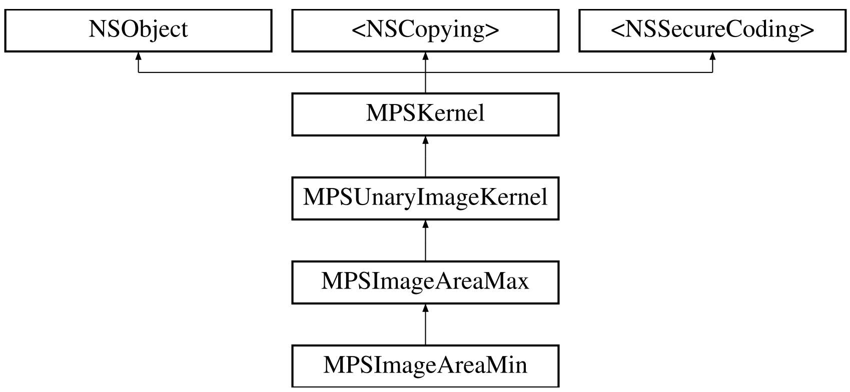
The documentation for this protocol was generated from the following file:

- MPSCore.framework/Headers/MPSImage.h

5.73 MPSImageAreaMax Class Reference

```
#import <MPSImageMorphology.h>
```

Inheritance diagram for MPSImageAreaMax:



Instance Methods

- (nonnull instancetype) - initWithDevice:kernelWidth:kernelHeight:
- (nullable instancetype) - initWithCoder:device:
- (nonnull instancetype) - initWithDevice:

Properties

- NSUInteger kernelHeight
- NSUInteger kernelWidth

Additional Inherited Members

5.73.1 Detailed Description

MPSImageMorphology.h MetalPerformanceShaders

Copyright

The MPSImageAreaMax kernel finds the maximum pixel value in a rectangular region centered around each pixel in the source image. If there are multiple channels in the source image, each channel is processed independently. The edgeMode property is assumed to always be MPSImageEdgeModeClamp for this filter.

5.73.2 Method Documentation

5.73.2.1 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSUnaryImageKernel.

5.73.2.2 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

a pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSUnaryImageKernel.

5.73.2.3 initWithDevice:kernelWidth:kernelHeight:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            kernelWidth:(NSUInteger) kernelWidth
            kernelHeight:(NSUInteger) kernelHeight
```

Set the kernel height and width

Parameters

device	The device the filter will run on
kernelWidth	The width of the kernel. Must be an odd number.
kernelHeight	The height of the kernel. Must be an odd number.

5.73.3 Property Documentation

5.73.3.1 kernelHeight

```
- kernelHeight [read], [nonatomic], [assign]
```

The height of the filter window. Must be an odd number.

5.73.3.2 kernelWidth

```
- kernelWidth [read], [nonatomic], [assign]
```

The width of the filter window. Must be an odd number.

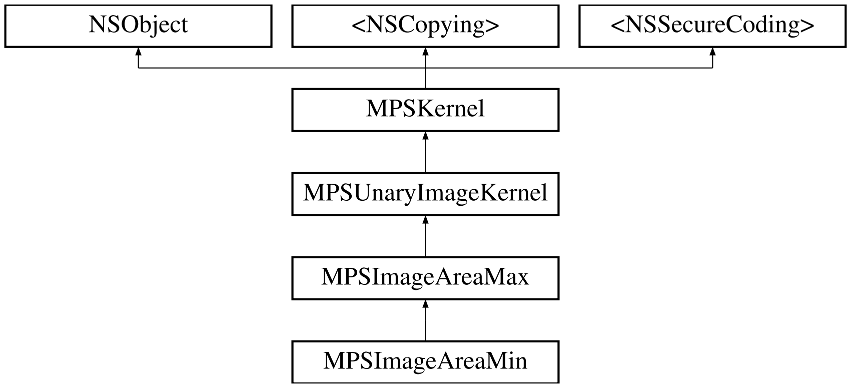
The documentation for this class was generated from the following file:

- MPSImageMorphology.h

5.74 MPSImageAreaMin Class Reference

```
#import <MPSImageMorphology.h>
```

Inheritance diagram for MPSImageAreaMin:



Additional Inherited Members

5.74.1 Detailed Description

The MPSImageAreaMin finds the minimum pixel value in a rectangular region centered around each pixel in the source image. If there are multiple channels in the source image, each channel is processed independently. It has the same methods as MPSImageAreaMax The edgeMode property is assumed to always be MPSImageEdge←ModeClamp for this filter.

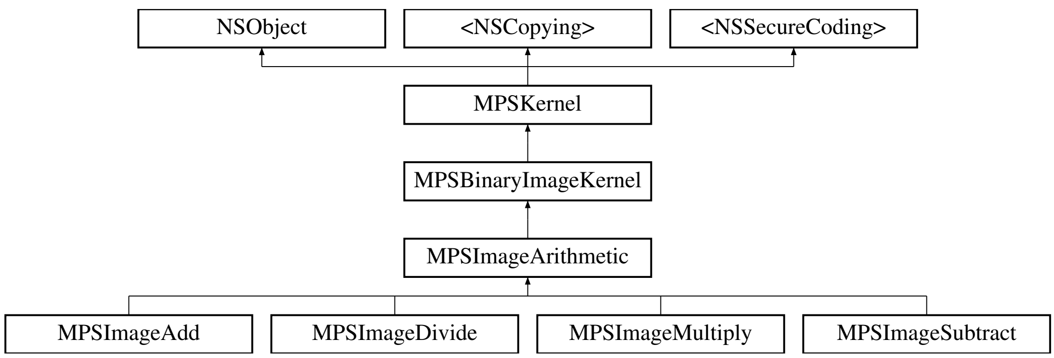
The documentation for this class was generated from the following file:

- MPSImageMorphology.h

5.75 MPSImageArithmetic Class Reference

```
#import <MPSImageMath.h>
```

Inheritance diagram for MPSImageArithmetic:



Instance Methods

- (nonnull instancetype) - initWithDevice:

Properties

- float primaryScale
- float secondaryScale
- float bias
- NSUInteger secondaryStrideInPixelsX
- NSUInteger secondaryStrideInPixelsY
- MTLSize primaryStrideInPixels
- MTLSize secondaryStrideInPixels

Additional Inherited Members

5.75.1 Detailed Description

MPSImageMath.h MetalPerformanceShaders.framework

Copyright

 MetalPerformanceShaders math filters

This depends on Metal.framework. This filter takes two source images, a primary source image and a secondary source image, and outputs a single destination image. It applies an element-wise arithmetic operator to each pixel in a primary source image and a corresponding pixel in a secondary source image over a specified region.

The supported arithmetic operators are the following:

- Addition
- Subtraction
- Multiplication
- Division

This filter takes additional parameters: primaryScale, secondaryScale, and bias. The default value for primaryScale and secondaryScale is 1.0f. The default value for bias is 0.0f. This filter applies primaryScale, secondaryScale, and bias to the primary source pixel (x) and secondary source pixel (y) in the following way:

- Addition: result = ((primaryScale $*$ x) + (secondaryScale $*$ y)) + bias
- Subtraction: result = ((primaryScale $*$ x) - (secondaryScale $*$ y)) + bias
- Multiplicaton: result = ((primaryScale $*$ x) $*$ (secondaryScale $*$ y)) + bias
- Division: result = ((primaryScale $*$ x) / (secondaryScale $*$ y)) + bias

This filter also takes the following additional parameters:

- primaryStrideInPixels
- secondaryStrideInPixels These parameters can be used to control broadcasting for the data stored in the primary and secondary source images. For example, setting all strides for the primary source image to 0 will result in the primarySource image being treated as a scalar value. The default value of these parameters is 1.

This filter accepts uint and int data in addition to unorm and floating-point data.

You must use one of the sub-classes of MPSImageArithmetic.

5.75.2 Method Documentation

5.75.2.1 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

a pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSBinaryImageKernel.

Reimplemented in MPSImageAdd, MPSImageSubtract, MPSImageMultiply, and MPSImageDivide.

5.75.3 Property Documentation

5.75.3.1 bias

```
- (float) bias  [read], [write], [nonatomic], [assign]
```

5.75.3.2 primaryScale

```
- (float) primaryScale  [read], [write], [nonatomic], [assign]
```

5.75.3.3 primaryStrideInPixels

```
- primaryStrideInPixels  [read], [write], [nonatomic], [assign]
```

The secondarySource stride in the x, y, and z dimensions. The default value for each dimension is 1.

5.75.3.4 secondaryScale

```
- (float) secondaryScale [read], [write], [nonatomic], [assign]
```

5.75.3.5 secondaryStrideInPixels

```
- secondaryStrideInPixels [read], [write], [nonatomic], [assign]
```

The secondarySource stride in the x, y, and z dimensions. The default value for each dimension is 1.

5.75.3.6 secondaryStrideInPixelsX

```
- secondaryStrideInPixelsX [read], [write], [nonatomic], [assign]
```

The secondarySource stride in the x dimension. The default value is 1.

5.75.3.7 secondaryStrideInPixelsY

```
- secondaryStrideInPixelsY [read], [write], [nonatomic], [assign]
```

The secondarySource stride in the y dimension. The default value is 1.

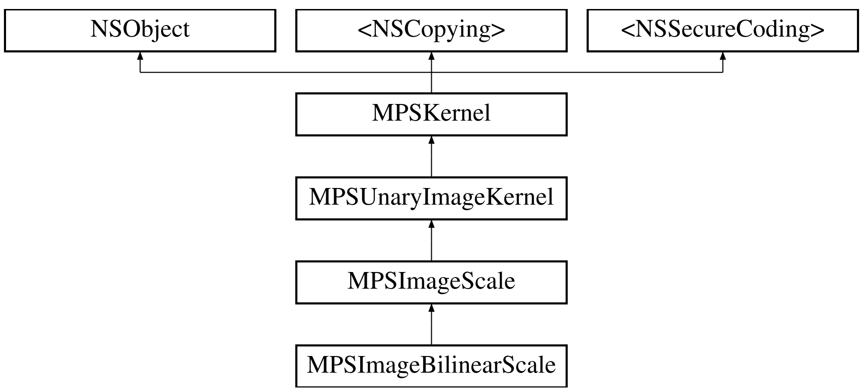
The documentation for this class was generated from the following file:

- MPSImageMath.h

5.76 MPSImageBilinearScale Class Reference

```
#import <MPSImageResampling.h>
```

Inheritance diagram for MPSImageBilinearScale:



Instance Methods

- (nonnull instancetype) - initWithDevice:
- (nullable instancetype) - initWithCoder:device:

Additional Inherited Members

5.76.1 Detailed Description

Resize an image and / or change its aspect ratio The MPSImageBilinearScale filter can be used to resample an existing image using a bilinear filter. This is typically used to reduce the size of an image.

5.76.2 Method Documentation

5.76.2.1 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSImageScale.

5.76.2.2 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

a pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSImageScale.

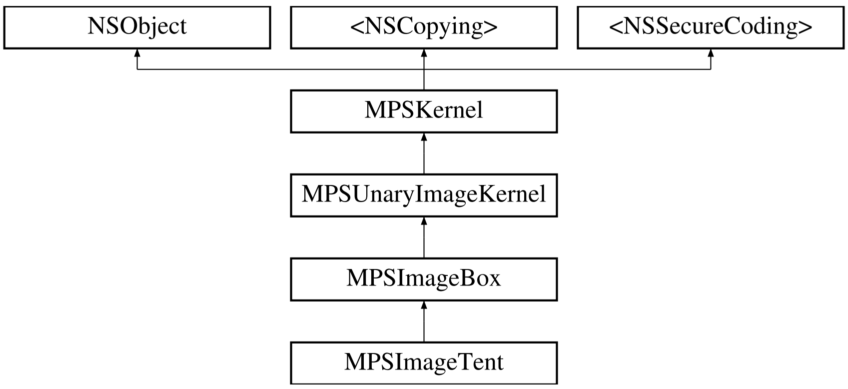
The documentation for this class was generated from the following file:

- MPSImageResampling.h

5.77 MPSImageBox Class Reference

```
#import <MPSImageConvolution.h>
```

Inheritance diagram for MPSImageBox:



Instance Methods

- (nonnull instancetype) - initWithDevice:kernelWidth:kernelHeight:
- (nullable instancetype) - initWithCoder:device:
- (nonnull instancetype) - initWithDevice:

Properties

- NSUInteger kernelHeight
- NSUInteger kernelWidth

Additional Inherited Members

5.77.1 Detailed Description

The MPSImageBox convolves an image with given filter of odd width and height. The kernel elements all have equal weight, achieving a blur effect. (Each result is the unweighted average of the surrounding pixels.) This allows for much faster algorithms, espcially for larger blur radii. The box height and width must be odd numbers. The box blur is a separable filter. The implementation is aware of this and will act accordingly to give best performance for multi-dimensional blurs.

5.77.2 Method Documentation

5.77.2.1 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSUnaryImageKernel.

5.77.2.2 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

a pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSUnaryImageKernel.

5.77.2.3 initWithDevice:kernelWidth:kernelHeight:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            kernelWidth:(NSUInteger) kernelWidth
            kernelHeight:(NSUInteger) kernelHeight
```

Initialize a filter for a particular kernel size and device

Parameters

device	The device the filter will run on
kernelWidth	the width of the kernel. Must be an odd number.
kernelHeight	the height of the kernel. Must be an odd number.

Returns

A valid object or nil, if failure.

5.77.3 Property Documentation

5.77.3.1 kernelHeight

```
- kernelHeight [read], [nonatomic], [assign]
```

The height of the filter window.

5.77.3.2 kernelWidth

```
- kernelWidth [read], [nonatomic], [assign]
```

The width of the filter window.

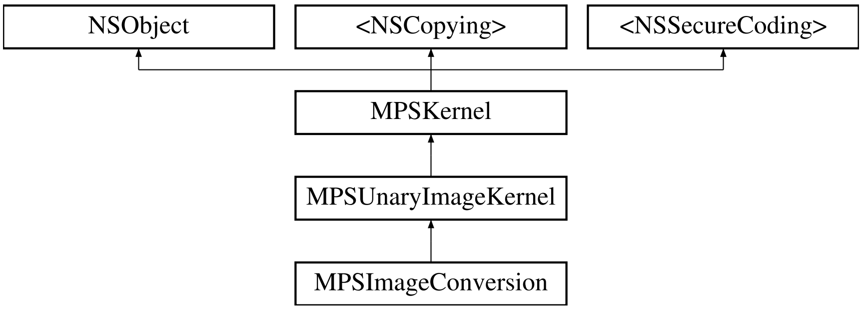
The documentation for this class was generated from the following file:

- MPSImageConvolution.h

5.78 MPSImageConversion Class Reference

```
#import <MPSImageConversion.h>
```

Inheritance diagram for MPSImageConversion:



Instance Methods

- (nonnull instancetype) - initWithDevice:srcAlpha:destAlpha:backgroundColor:conversionInfo:

Properties

- MPSAlphaType sourceAlpha
- MPSAlphaType destinationAlpha

Additional Inherited Members

5.78.1 Detailed Description

MPSImageConversions.h MetalPerformanceShaders.framework

Copyright

 MetalPerformanceShaders conversion filters MPS_CLAS↩
S_AVAILABLE_STARTING

The MPSImageConversion filter performs a conversion from source to destination

5.78.2 Method Documentation

5.78.2.1 initWithDevice:srcAlpha:destAlpha:backgroundColor:conversionInfo:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            srcAlpha:(MPSAlphaType) srcAlpha
            destAlpha:(MPSAlphaType) destAlpha
            backgroundColor:(nullable CGFloat *) backgroundColor
            conversionInfo:(nullable CGColorConversionInfoRef) conversionInfo
```

Create a converter that can convert texture colorspace, alpha and texture format Create a converter that can convert texture colorspace, alpha and MTLPixelFormat. Optimized cases exist for NULL color space converter and no alpha conversion.

Parameters

device	The device the filter will run on
srcAlpha	The alpha encoding for the source texture
destAlpha	The alpha encoding for the destination texture
backgroundColor	An array of CGFloats giving the background color to use when flattening an image. The color is in the source colorspace. The length of the array is the number of color channels in the src colorspace. If NULL, use {0}.
conversionInfo	The colorspace conversion to use. May be NULL, indicating no color space conversions need to be done.

Returns

An initialized MPSImageConversion object.

5.78.3 Property Documentation

5.78.3.1 destinationAlpha

```
- destinationAlpha  [read], [nonatomic], [assign]
```

Premultiplication description for the destinationAlpha texture Colorspace conversion operations produce non-premultiplied data. Use this property to tag cases where premultiplied results are required. If MPSPixelAlpha↩_AlphaIsOne is used, the alpha channel will be set to 1. Default: MPSPixelAlpha_AlphaIsOne

5.78.3.2 sourceAlpha

```
- sourceAlpha  [read], [nonatomic], [assign]
```

Premultiplication description for the source texture Most colorspace conversion operations can not work directly on premultiplied data. Use this property to tag premultiplied data so that the source texture can be unpremultiplied prior to application of these transforms. Default: MPSPixelAlpha_AlphaIsOne

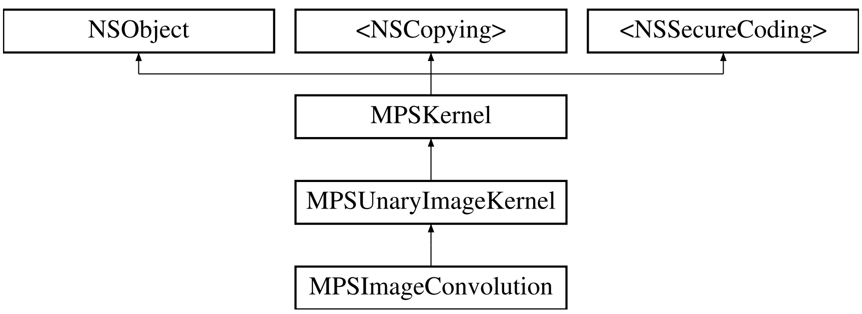
The documentation for this class was generated from the following file:

- MPSImageConversion.h

5.79 MPSImageConvolution Class Reference

```
#import <MPSImageConvolution.h>
```

Inheritance diagram for MPSImageConvolution:



Instance Methods

- (nonnull instancetype) - initWithDevice:kernelWidth:kernelHeight:weights:
- (nullable instancetype) - initWithCoder:device:

Properties

- NSUInteger kernelHeight
- NSUInteger kernelWidth
- float bias

Additional Inherited Members

5.79.1 Detailed Description

MPSImageConvolution.h MetalPerformanceShaders

Copyright

MetalPerformanceShaders Convolution Filters

The MPSImageConvolution convolves an image with given filter of odd width and height. The center of the kernel aligns with the MPSImageConvolution.offset. That is, the position of the top left corner of the area covered by the kernel is given by MPSImageConvolution.offset - {kernel_width>>1, kernel_height>>1, 0}

Optimized cases include 3x3,5x5,7x7,9x9,11x11, 1xN and Nx1. If a convolution kernel does not fall into one of these cases but is a rank-1 matrix (a.k.a. separable) then it will fall on an optimzied separable path. Other convolutions will execute with full MxN complexity.

If there are multiple channels in the source image, each channel is processed independently.

Separable convolution filters may perform better when done in two passes. A convolution filter is separable if the ratio of filter values between all rows is constant over the whole row. For example, this edge detection filter:

```
-1      0       1
-2      0       2
-1      0       1
```

can be separated into the product of two vectors:

```
1
2       x     [-1   0    1]
1
```

and consequently can be done as two, one-dimensional convolution passes back to back on the same image. In this way, the number of multiplies (ignoring the fact that we could skip zeros here) is reduced from $3*3=9$ to $3+3 = 6$. There are similar savings for addition. For large filters, the savings can be profound.

5.79.2 Method Documentation

5.79.2.1 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
                (NSCoder *__nonnull) aDecoder
                device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSUnaryImageKernel.

5.79.2.2 initWithDevice:kernelWidth:kernelHeight:weights:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            kernelWidth:(NSUInteger) kernelWidth
            kernelHeight:(NSUInteger) kernelHeight
            weights:(const float *__nonnull) kernelWeights
```

Initialize a convolution filter

Parameters

device	The device the filter will run on
kernelWidth	the width of the kernel
kernelHeight	the height of the kernel
kernelWeights	A pointer to an array of kernelWidth ∗ kernelHeight values to be used as the kernel. These are in row major order.

Returns

A valid MPSImageConvolution object or nil, if failure.

5.79.3 Property Documentation

5.79.3.1 bias

```
- bias  [read], [write], [nonatomic], [assign]
```

The bias is a value to be added to convolved pixel before it is converted back to the storage format. It can be used to convert negative values into a representable range for a unsigned MTLPixelFormat. For example, many edge detection filters produce results in the range [-k,k]. By scaling the filter weights by 0.5/k and adding 0.5, the results will be in range [0,1] suitable for use with unorm formats. It can be used in combination with renormalization of the filter weights to do video ranging as part of the convolution effect. It can also just be used to increase the brightness of the image.

Default value is 0.0f.

5.79.3.2 kernelHeight

```
- kernelHeight [read], [nonatomic], [assign]
```

The height of the filter window. Must be an odd number.

5.79.3.3 kernelWidth

```
- kernelWidth [read], [nonatomic], [assign]
```

The width of the filter window. Must be an odd number.

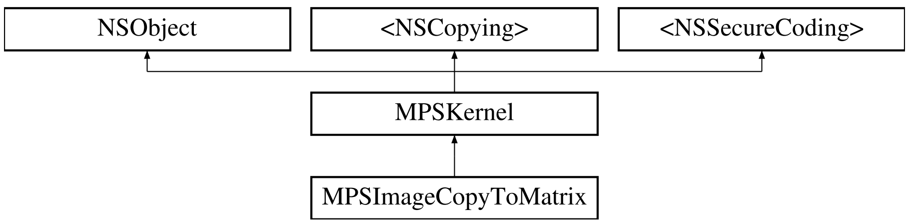
The documentation for this class was generated from the following file:

- MPSImageConvolution.h

5.80 MPSImageCopyToMatrix Class Reference

```
#import <MPSImageCopy.h>
```

Inheritance diagram for MPSImageCopyToMatrix:



Instance Methods

- (nonnull instancetype) - initWithDevice:dataLayout:
- (nullable instancetype) - initWithCoder:device:
- (void) - encodeToCommandBuffer:sourceImage:destinationMatrix:

Properties

- MTLOrigin destinationMatrixOrigin
- NSUInteger destinationMatrixBatchIndex
- MPSDataLayout dataLayout

Additional Inherited Members

5.80.1 Detailed Description

MPSImageCopy.h MetalPerformanceShaders.framework

Copyright

Copyright (c) 2017 Apple Inc. All rights reserved. MetalPerformanceShaders histogram filters

The MPSImageCopyToMatrix copies image data to a MPSMatrix. The image data is stored in a row of a matrix. The dataLayout specifies the order in which the feature channels in the MPSImage get stored in the matrix. If MPSImage stores a batch of images, the images are copied into multiple rows, one row per image.

The number of elements in a row in the matrix must be >= image width ∗ image height ∗ number of featureChannels in the image.

5.80.2 Method Documentation

5.80.2.1 encodeToCommandBuffer:sourceImage:destinationMatrix:()

```
- (void) encodeToCommandBuffer:
            (nonnull id< MTLCommandBuffer >) commandBuffer
            sourceImage:(nonnull MPSImage ∗) sourceImage
            destinationMatrix:(nonnull MPSMatrix ∗) destinationMatrix
```

Encode a kernel that copies a MPSImage to a MPSMatrix into a command buffer using a MTLComputeCommand←Encoder. The kernel copies feature channels from sourceImage to the buffer associated with destinationMatrix. The kernel will not begin to execute until after the command buffer has been enqueued and committed.

NOTE: The destinationMatrix.dataType must match the feature channel data type in sourceImage.

Parameters

commandBuffer	A valid MTLCommandBuffer.
sourceImage	A valid MPSImage describing the image to copy from.
destinationMatrix	A valid MPSMatrix or MPSTemporaryMatrix object describing the matrix to copy to.

5.80.2.2 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder ∗__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSKernel.

5.80.2.3 initWithDevice:dataLayout:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            dataLayout:(MPSDataLayout) dataLayout
```

Initialize a MPSMatrixCopy object on a device

Parameters

device	The device the kernel will run on
dataLayout	The data layout

Returns

A valid MPSMatrixCopy object or nil, if failure.

5.80.3 Property Documentation

5.80.3.1 dataLayout

```
- dataLayout  [read], [nonatomic], [assign]
```

The data layout to use Returns the data layout. When copying from a MPSImage to a MPSMatrix, this describes the order in which the image values are stored in the buffer associated with the MPSMatrix. Default: MPSDataLayoutFeatureChannelsxHeightxWidth

5.80.3.2 destinationMatrixBatchIndex

```
- destinationMatrixBatchIndex  [read], [write], [nonatomic], [assign]
```

The index of the destination matrix in the batch. This property is modifiable and defaults to 0 at initialization time.

5.80.3.3 destinationMatrixOrigin

```
- destinationMatrixOrigin  [read], [write], [nonatomic], [assign]
```

The origin, relative to [0, 0] in the destination matrix, at which to start writing results. This property is modifiable and defaults to [0, 0] at initialization time. If a different origin is desired then this should be modified prior to encoding the kernel. The z value must be 0.

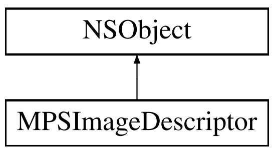
The documentation for this class was generated from the following file:

- MPSImageCopy.h

5.81 MPSImageDescriptor Class Reference

```
#import <MPSImage.h>
```

Inheritance diagram for MPSImageDescriptor:



Class Methods

- (__nonnull instancetype) + imageDescriptorWithChannelFormat:width:height:featureChannels:
- (__nonnull instancetype) + imageDescriptorWithChannelFormat:width:height:featureChannels:numberOf↩ Images:usage:

Properties

- NSUInteger width
- NSUInteger height
- NSUInteger featureChannels
- NSUInteger numberOfImages
- MTLPixelFormat pixelFormat
- MPSImageFeatureChannelFormat channelFormat
- MTLCPUCacheMode cpuCacheMode
- MTLStorageMode storageMode
- MTLTextureUsage usage

5.81.1 Detailed Description

MPSImage.h MPSCore.framework

Copyright

Copyright (c) 2015-2017 Apple Inc. All rights reserved. A MPSImage is a MTLTexture abstraction that allows for more than 4 channels, and for temporary images.

This depends on Metal.framework A MPSImageDescriptor object describes a attributes of MPSImage and is used to create one (see MPSImage discussion below)

5.81.2 Method Documentation

5.81.2.1 imageDescriptorWithChannelFormat:width:height:featureChannels:()

```
+ (__nonnull instancetype) imageDescriptorWithChannelFormat:
            (MPSImageFeatureChannelFormat) channelFormat
            width:(NSUInteger) width
            height:(NSUInteger) height
            featureChannels:(NSUInteger) featureChannels
```

Create a MPSImageDescriptor for a single read/write cnn image.

5.81.2.2 imageDescriptorWithChannelFormat:width:height:featureChannels:numberOfImages:usage:()

```
+ (__nonnull instancetype) imageDescriptorWithChannelFormat:
            (MPSImageFeatureChannelFormat) channelFormat
            width:(NSUInteger) width
            height:(NSUInteger) height
            featureChannels:(NSUInteger) featureChannels
            numberOfImages:(NSUInteger) numberOfImages
            usage:(MTLTextureUsage) usage
```

Create a MPSImageDescriptor for a read/write cnn image with option to set usage and batch size (numberOf↩ Images).

5.81.3 Property Documentation

5.81.3.1 channelFormat

```
- channelFormat [read], [write], [nonatomic], [assign]
```

The storage format to use for each channel in the image.

5.81.3.2 cpuCacheMode

```
- cpuCacheMode [read], [write], [nonatomic], [assign]
```

Options to specify CPU cache mode of texture resource. Default = MTLCPUCacheModeDefaultCache

5.81.3.3 featureChannels

```
- featureChannels [read], [write], [nonatomic], [assign]
```

The number of feature channels per pixel. Default = 1.

5.81.3.4 height

```
- height [read], [write], [nonatomic], [assign]
```

The height of the CNN image. The formal height of the CNN image in pixels. Default = 1.

5.81.3.5 numberOfImages

```
- numberOfImages [read], [write], [nonatomic], [assign]
```

The number of images for batch processing. Default = 1.

5.81.3.6 pixelFormat

```
- pixelFormat [read], [nonatomic], [assign]
```

The MTLPixelFormat expected for the underlying texture.

5.81.3.7 storageMode

```
- storageMode [read], [write], [nonatomic], [assign]
```

To specify storage mode of texture resource. Storage mode options:

```
Default =   MTLStorageModeShared on iOS
            MTLStorageModeManaged on Mac OSX
MTLStorageModeShared not supported on Mac OSX.
See Metal headers for synchronization requirements when using StorageModeManaged
```

5.81.3.8 usage

```
- usage [read], [write], [nonatomic], [assign]
```

Description of texture usage. Default = MTLTextureUsageShaderRead/Write

5.81.3.9 width

- width [read], [write], [nonatomic], [assign]

The width of the CNN image. The formal width of the CNN image in pixels. Default = 1.

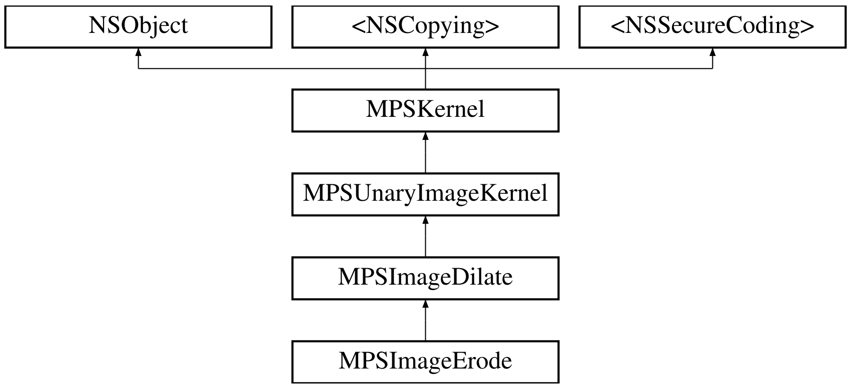
The documentation for this class was generated from the following file:

- MPSCore.framework/Headers/MPSImage.h

5.82 MPSImageDilate Class Reference

#import <MPSImageMorphology.h>

Inheritance diagram for MPSImageDilate:



Instance Methods

- (nonnull instancetype) - initWithDevice:kernelWidth:kernelHeight:values:
- (nonnull instancetype) - initWithDevice:
- (nullable instancetype) - initWithCoder:device:

Properties

- NSUInteger kernelHeight
- NSUInteger kernelWidth

Additional Inherited Members

5.82.1 Detailed Description

The MPSImageDilate finds the maximum pixel value in a rectangular region centered around each pixel in the source image. It is like the MPSImageAreaMax, except that the intensity at each position is calculated relative to a different value before determining which is the maximum pixel value, allowing for shaped, non-rectangular morphological probes.

```
for each pixel in the filter window:
    value =  pixel[filterY][filterX] - filter[filterY*filter_width+filterX]
    if( value > bestValue ){
         result = value
         bestValue = value;
    }
```

A filter that contains all zeros and is identical to a MPSImageAreaMax filter. The center filter element is assumed to be 0 to avoid causing a general darkening of the image.

The edgeMode property is assumed to always be MPSImageEdgeModeClamp for this filter.

5.82.2 Method Documentation

5.82.2.1 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSUnaryImageKernel.

5.82.2.2 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

a pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSUnaryImageKernel.

5.82.2.3 initWithDevice:kernelWidth:kernelHeight:values:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            kernelWidth:(NSUInteger) kernelWidth
            kernelHeight:(NSUInteger) kernelHeight
            values:(const float *__nonnull) values
```

Init a object with kernel height, width and weight values. Each dilate shape probe defines a 3D surface of values. These are arranged in order left to right, then top to bottom in a 1D array. (values[kernelWidth*y+x] = probe[y][x]) Values should be generally be in the range [0,1] with the center pixel tending towards 0 and edges towards 1. However, any numerical value is allowed. Calculations are subject to the usual floating-point rounding error.

Parameters

device	The device the filter will run on
kernelWidth	The width of the kernel. Must be an odd number.
kernelHeight	The height of the kernel. Must be an odd number.
values	The set of values to use as the dilate probe. The values are copied into the filter. To avoid image ligthening or darkening, the center value should be 0.0f.

5.82.3 Property Documentation

5.82.3.1 kernelHeight

```
- kernelHeight [read], [nonatomic], [assign]
```

The height of the filter window. Must be an odd number.

5.82.3.2 kernelWidth

```
- kernelWidth [read], [nonatomic], [assign]
```

The width of the filter window. Must be an odd number.

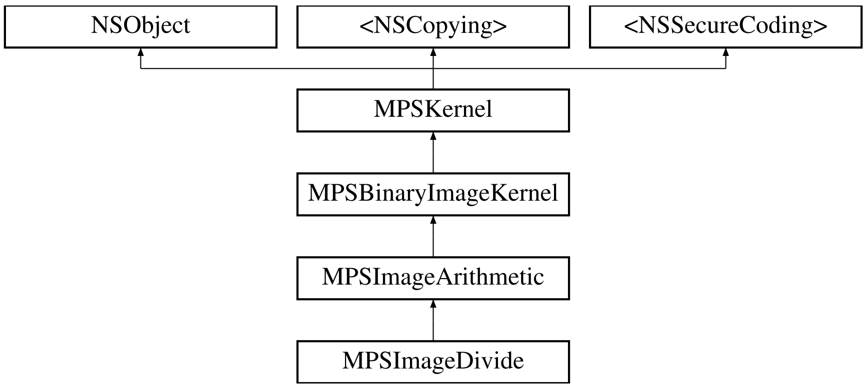
The documentation for this class was generated from the following file:

- MPSImageMorphology.h

5.83 MPSImageDivide Class Reference

```
#import <MPSImageMath.h>
```

Inheritance diagram for MPSImageDivide:



Instance Methods

- (nonnull instancetype) - initWithDevice:

Additional Inherited Members

5.83.1 Detailed Description

This depends on Metal.framework. Specifies the division operator. For each pixel in the primary source image (x) and each pixel in a secondary source image (y), it applies the following function: result = ((primaryScale $*$ x) / (secondaryScale $*$ y)) + bias.

5.83.2 Method Documentation

5.83.2.1 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Initialize the division operator

Parameters

device	The device the filter will run on.

Returns

A valid MPSImageDivide object or nil, if failure.

Reimplemented from MPSImageArithmetic.

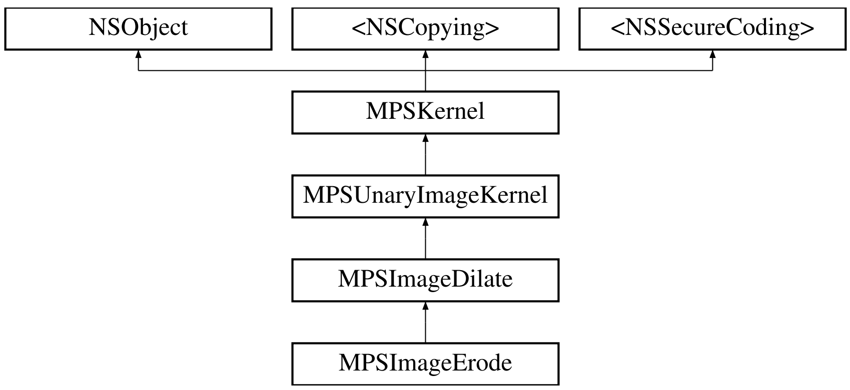
The documentation for this class was generated from the following file:

- MPSImageMath.h

5.84 MPSImageErode Class Reference

```
#import <MPSImageMorphology.h>
```

Inheritance diagram for MPSImageErode:



Additional Inherited Members

5.84.1 Detailed Description

The MPSImageErode filter finds the minimum pixel value in a rectangular region centered around each pixel in the source image. It is like the MPSImageAreaMin, except that the intensity at each position is calculated relative to a different value before determining which is the maximum pixel value, allowing for shaped, non-rectangular morphological probes.

```
for each pixel in the filter window:
    value =  pixel[filterY][filterX] + filter[filterY*filter_width+filterX]
    if( value < bestValue ){
         result = value
         bestValue = value;
    }
```

A filter that contains all zeros is identical to a MPSImageAreaMin filter. The center filter element is assumed to be 0, to avoid causing a general lightening of the image.

The definition of the filter for MPSImageErode is different from vImage. (MPSErode_filter_value = 1.0f-vImageErode_filter_value.) This allows MPSImageDilate and MPSImageErode to use the same filter, making open and close operators easier to write. The edgeMode property is assumed to always be MPSImageEdgeModeClamp for this filter.

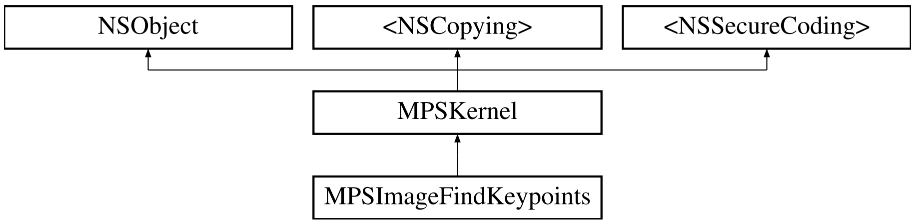
The documentation for this class was generated from the following file:

- MPSImageMorphology.h

5.85 MPSImageFindKeypoints Class Reference

```
#import <MPSImageKeypoint.h>
```

Inheritance diagram for MPSImageFindKeypoints:



Instance Methods

- (nonnull instancetype) - initWithDevice:info:
- (nonnull instancetype) - initWithDevice:
- (nullable instancetype) - initWithCoder:device:
- (void) - encodeToCommandBuffer:sourceTexture:regions:numberOfRegions:keypointCountBuffer:keypointCountBufferOffset:keypointDataBuffer:keypointDataBufferOffset:

Properties

- MPSImageKeypointRangeInfo keypointRangeInfo

Additional Inherited Members

5.85.1 Detailed Description

The MPSImageFindKeypoints kernel is used to find a list of keypoints whose values are >= minimumPixelThresholdValue in MPSImageKeypointRangeInfo. The keypoints are generated for a specified region in the image. The pixel format of the source image must be MTLPixelFormatR8Unorm.

5.85.2 Method Documentation

5.85.2.1 encodeToCommandBuffer:sourceTexture:regions:numberOfRegions:keypointCountBuffer:keypointCountBufferOffset:keypointDataBuffer:keypointDataBufferOffset:()

```
- (void) encodeToCommandBuffer:
            (nonnull id< MTLCommandBuffer >) commandBuffer
            sourceTexture:(nonnull id< MTLTexture >) source
            regions:(const MTLRegion *__nonnull) regions
            numberOfRegions:(NSUInteger) numberOfRegions
            keypointCountBuffer:(nonnull id< MTLBuffer >) keypointCountBuffer
            keypointCountBufferOffset:(NSUInteger) keypointCountBufferOffset
            keypointDataBuffer:(nonnull id< MTLBuffer >) keypointDataBuffer
            keypointDataBufferOffset:(NSUInteger) keypointDataBufferOffset
```

Encode the filter to a command buffer using a MTLComputeCommandEncoder. The filter will not begin to execute until after the command buffer has been enqueued and committed.

Parameters

commandBuffer	A valid MTLCommandBuffer.
source	A valid MTLTexture containing the source image for the filter.
regions	An array of rectangles that describe regions in the image. The list of keypoints is generated for each individual rectangle specifed.
keypointCountBuffer	The list of keypoints for each specified region
keypointCountBufferOffset	Byte offset into keypointCountBufferOffset buffer at which to write the keypoint results. Must be a multiple of 32 bytes.
keypointDataBuffer	A valid MTLBuffer to receive the keypoint data results for each rectangle. The keypoint data for keypoints in each rectangle are stored consecutively. The keypoint data for each rectangle starts at the following offset: MPSImageKeypointRangeInfo.maximumKeyPoints ∗ rectangle index
keypointDataBufferOffset	Byte offset into keypointData buffer at which to write the keypoint results. Must be a multiple of 32 bytes.

5.85.2.2 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSKernel.

5.85.2.3 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

a pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSKernel.

5.85.2.4 initWithDevice:info:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            info:(const MPSImageKeypointRangeInfo *__nonnull) info
```

Specifies information to find keypoints in an image.

Parameters

device	The device the filter will run on
info	Pointer to the MPSImageKeypointRangeInfo struct

Returns

A valid MPSImageFindKeypoints object or nil, if failure.

5.85.3 Property Documentation

5.85.3.1 keypointRangeInfo

```
- keypointRangeInfo [read], [nonatomic], [assign]
```

Return a structure describing the keypoint range info Returns a MPSImageKeypointRangeInfo structure

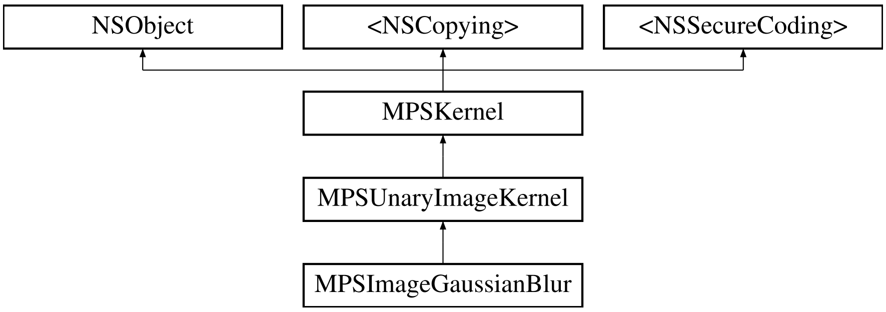
The documentation for this class was generated from the following file:

- MPSImageKeypoint.h

5.86 MPSImageGaussianBlur Class Reference

```
#import <MPSImageConvolution.h>
```

Inheritance diagram for MPSImageGaussianBlur:



Instance Methods

- (nonnull instancetype) - initWithDevice:sigma:
- (nullable instancetype) - initWithCoder:device:
- (nonnull instancetype) - initWithDevice:

Properties

- float sigma

Additional Inherited Members

5.86.1 Detailed Description

The MPSImageGaussianBlur convolves an image with gaussian of given sigma in both x and y direction.

```
The MPSImageGaussianBlur utilizes a very fast algorith that typically runs at approximately
1/2 of copy speeds. Notably, it is faster than either the tent or box blur except perhaps
for very large filter windows. Mathematically, it is an approximate gaussian. Some
non-gaussian behavior may be detectable with advanced analytical methods such as FFT.
If a analytically clean gaussian filter is required, please use the MPSImageConvolution
filter instead with an appropriate set of weights. The MPSImageGaussianBlur is intended
to be suitable for all common image processing needs demanding ~10 bits of precision or
less.
```

5.86.2 Method Documentation

5.86.2.1 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSUnaryImageKernel.

5.86.2.2 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

a pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSUnaryImageKernel.

5.86.2.3 initWithDevice:sigma:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            sigma:(float) sigma
```

Initialize a gaussian blur filter for a particular sigma and device

Parameters

device	The device the filter will run on
sigma	The standard deviation of gaussian blur filter. Gaussian weight, centered at 0, at integer grid i is given as w(i) = 1/sqrt(2∗pi∗sigma) ∗ exp(-i^2/2∗sigma^2) If we take cut off at 1% of w(0) (max weight) beyond which weights are considered 0, we have ceil (sqrt(-log(0.01)∗2)∗sigma) ∼ ceil(3.7∗sigma) as rough estimate of filter width

Returns

A valid object or nil, if failure.

5.86.3 Property Documentation

5.86.3.1 sigma

```
- sigma  [read], [nonatomic], [assign]
```

Read-only sigma value with which filter was created

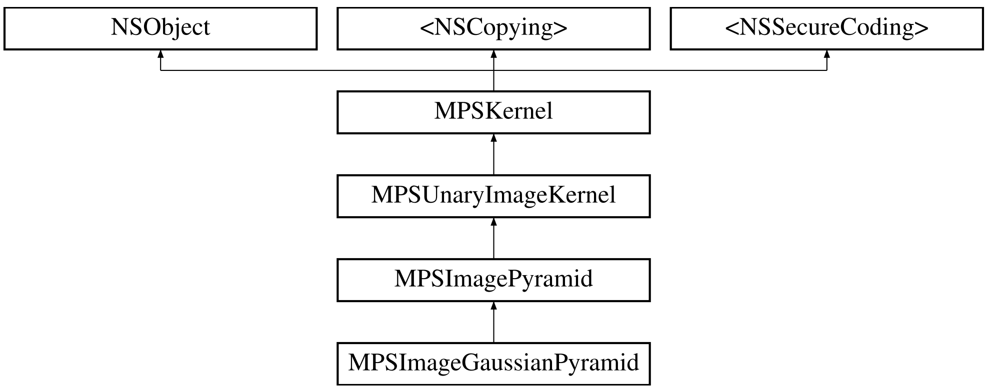
The documentation for this class was generated from the following file:

- MPSImageConvolution.h

5.87 MPSImageGaussianPyramid Class Reference

```
#import <MPSImageConvolution.h>
```

Inheritance diagram for MPSImageGaussianPyramid:



Additional Inherited Members

5.87.1 Detailed Description

The Gaussian image pyramid is constructed as follows: First the zeroth level mipmap of the input image is filtered with the specified convolution kernel. The default the convolution filter kernel is

```
k = w w^T, where w = [ 1/16,  1/4,  3/8,  1/4,  1/16 ]^T,
```

but the user may also tweak this kernel with a centerWeight parameter: 'a':

```
k = w w^T, where w = [ (1/4 - a/2),  1/4,  a,  1/4,  (1/4 - a/2) ]^T
```

or the user can provide a completely custom kernel. After this the image is downsampled by removing all odd rows and columns, which defines the next level in the Gaussian image pyramid. This procedure is continued until every mipmap level present in the image texture are filled with the pyramid levels.

In case of the Gaussian pyramid the user must run the operation in-place using: inPlaceTexture:fallbackCopy←Allocator:, where the fallback allocator is ignored.

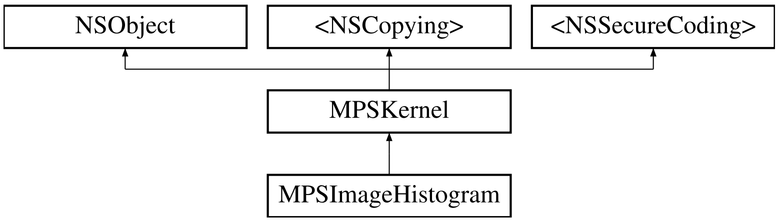
The documentation for this class was generated from the following file:

- MPSImageConvolution.h

5.88 MPSImageHistogram Class Reference

```
#import <MPSImageHistogram.h>
```

Inheritance diagram for MPSImageHistogram:



Instance Methods

- (nonnull instancetype) - initWithDevice:histogramInfo:
- (nullable instancetype) - initWithCoder:device:
- (void) - encodeToCommandBuffer:sourceTexture:histogram:histogramOffset:
- (size_t) - histogramSizeForSourceFormat:

Properties

- MTLRegion clipRectSource
- BOOL zeroHistogram
- vector_float4 minPixelThresholdValue
- MPSImageHistogramInfo histogramInfo

Additional Inherited Members

5.88.1 Detailed Description

The MPSImageHistogram computes the histogram of an image.

5.88.2 Method Documentation

5.88.2.1 encodeToCommandBuffer:sourceTexture:histogram:histogramOffset:()

```
- (void) encodeToCommandBuffer:
            (nonnull id< MTLCommandBuffer >) commandBuffer
            sourceTexture:(nonnull id< MTLTexture >) source
            histogram:(nonnull id< MTLBuffer >) histogram
            histogramOffset:(NSUInteger) histogramOffset
```

Encode the filter to a command buffer using a MTLComputeCommandEncoder. The filter will not begin to execute until after the command buffer has been enqueued and committed.

Parameters

commandBuffer	A valid MTLCommandBuffer.
source	A valid MTLTexture containing the source image for the filter
histogram	A valid MTLBuffer to receive the histogram results.
histogramOffset	Byte offset into histogram buffer at which to write the histogram results. Must be a multiple of 32 bytes. The histogram results / channel are stored together. The number of channels for which histogram results are stored is determined by the number of channels in the image. If histogramInfo.histogramForAlpha is false and the source image is RGBA then only histogram results for RGB channels are stored.

The histogram results are stored in the histogram buffer as follows:

- histogram results for the R channel for all bins followed by
- histogram results for the G channel for all bins followed by
- histogram results for the B channel for all bins followed by
- histogram results for the A channel for all bins

5.88.2.2 histogramSizeForSourceFormat:()

```
- (size_t) histogramSizeForSourceFormat:
            (MTLPixelFormat) sourceFormat
```

The amount of space in the output MTLBuffer the histogram will take up. This convenience function calculates the minimum amount of space needed in the output histogram for the results. The MTLBuffer should be at least this length, longer if histogramOffset is non-zero.

Parameters

sourceFormat	The MTLPixelFormat of the source image. This is the source parameter of -encodeToCommandBuffer: sourceTexture:histogram:histogramOffset

Returns

The number of bytes needed to store the result histograms.

5.88.2.3 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSKernel.

5.88.2.4 initWithDevice:histogramInfo:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            histogramInfo:(const MPSImageHistogramInfo *__nonnull) histogramInfo
```

Specifies information to compute the histogram for channels of an image.

Parameters

device	The device the filter will run on
histogramInfo	Pointer to the MPSHistogramInfo struct

Returns

A valid MPSImageHistogram object or nil, if failure.

5.88.3 Property Documentation

5.88.3.1 clipRectSource

```
- clipRectSource  [read], [write], [nonatomic], [assign]
```

The source rectangle to use when reading data. A MTLRegion that indicates which part of the source to read. If the clipRectSource does not lie completely within the source image, the intersection of the image bounds and clip↩RectSource will be used. The clipRectSource replaces the MPSUnaryImageKernel offset parameter for this filter. The latter is ignored. Default: MPSRectNoClip, use the entire source texture.

5.88.3.2 histogramInfo

```
- histogramInfo  [read], [nonatomic], [assign]
```

Return a structure describing the histogram content Returns a MPSImageHistogramInfo structure describing the format of the histogram.

5.88.3.3 minPixelThresholdValue

```
- minPixelThresholdValue [read], [write], [nonatomic], [assign]
```

The minimum pixel threshold value The histogram entries will be incremented only if pixel value is >= minPixelThresholdValue. The minPixelThresholdValue is a floating-point value. For unsigned normalized textures, the minPixelThresholdValue should be a value between 0.0f and 1.0f (for eg. MTLPixelFormatRGBA8Unorm). For signed normalized textures, the minPixelThresholdValue should be a value between -1.0f and 1.0f (for eg. MTLPixelFormatRGBA8Snorm). Default: vector_float4(0.0f).

5.88.3.4 zeroHistogram

```
- zeroHistogram [read], [write], [nonatomic], [assign]
```

Zero-initalize the histogram results Indicates that the memory region in which the histogram results are to be written in the histogram buffer are to be zero-initialized or not. Default: YES.

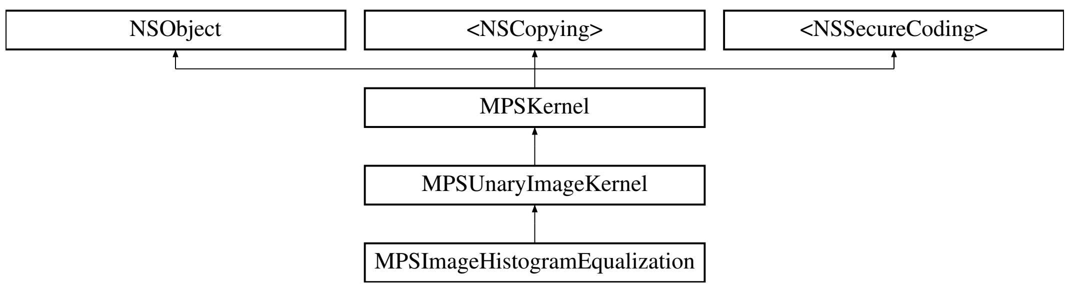
The documentation for this class was generated from the following file:

- MPSImageHistogram.h

5.89 MPSImageHistogramEqualization Class Reference

```
#import <MPSImageHistogram.h>
```

Inheritance diagram for MPSImageHistogramEqualization:



Instance Methods

- (nonnull instancetype) - initWithDevice:histogramInfo:
- (nullable instancetype) - initWithCoder:device:
- (void) - encodeTransformToCommandBuffer:sourceTexture:histogram:histogramOffset:

Properties

- MPSImageHistogramInfo histogramInfo

Additional Inherited Members

5.89.1 Detailed Description

The MPSImageHistogramEqualization performs equalizes the histogram of an image. The process is divided into three steps.

1. Call -initWithDevice:histogramInfo: This creates a MPSImageHistogramEqualization object. It is done when the method returns.
2. Call -encodeTransform:sourceTexture:histogram:histogramOffset: This creates a privately held image transform (i.e. a cumulative distribution function of the histogram) which will be used to equalize the distribution of the histogram of the source image. This process runs on a MTLCommandBuffer when it is committed to a MTLCommandQueue. It must complete before the next step can be run. It may be performed on the same MTLCommandBuffer. The histogram argument specifies the histogram buffer which contains the histogram values for sourceTexture. The sourceTexture argument is used by encodeTransform to determine the number of channels and therefore which histogram data in histogram buffer to use. The histogram for sourceTexture must have been computed either on the CPU or using the MPSImageHistogram kernel
3. Call -encodeToCommandBuffer:sourceTexture:destinationTexture: to read data from sourceTexture, apply the equalization transform to it and write to destination texture. This step is also done on the GPU on a MTL↩ CommandQueue.

You can reuse the same equalization transform on other images to perform the same transform on those images. (Since their distribution is probably different, they will probably not be equalized by it.) This filter usually will not be able to work in place.

5.89.2 Method Documentation

5.89.2.1 encodeTransformToCommandBuffer:sourceTexture:histogram:histogramOffset:()

```
- (void) encodeTransformToCommandBuffer:
            (nonnull id< MTLCommandBuffer >) commandBuffer
            sourceTexture:(nonnull id< MTLTexture >) source
            histogram:(nonnull id< MTLBuffer >) histogram
            histogramOffset:(NSUInteger) histogramOffset
```

Encode the transform function to a command buffer using a MTLComputeCommandEncoder. The transform function computes the equalization lookup table. The transform function will not begin to execute until after the command buffer has been enqueued and committed. This step will need to be repeated with the new MPSKernel if -copy↩ WithZone:device or -copyWithZone: is called. The transform is stored as internal state to the object. You still need to call -encodeToCommandBuffer:sourceTexture:destinationTexture: afterward to apply the transform to produce a result texture.

Parameters

commandBuffer	A valid MTLCommandBuffer.
source	A valid MTLTexture containing the source image for the filter.
histogram	A valid MTLBuffer containing the histogram results for an image. This filter will use these histogram results to generate the cumulative histogram for equalizing the image. The histogram results / channel are stored together. The number of channels for which histogram results are stored is determined by the number of channels in the image. If histogramInfo.histogramForAlpha is false and the source image is RGBA then only histogram results for RGB channels are stored.
histogramOffset	A byte offset into the histogram MTLBuffer where the histogram starts. Must conform to alignment requirements for [MTLComputeCommandEncoder setBuffer:offset:atIndex:] offset

5.89.2.2 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSUnaryImageKernel.

5.89.2.3 initWithDevice:histogramInfo:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            histogramInfo:(const MPSImageHistogramInfo *__nonnull) histogramInfo
```

Specifies information about the histogram for the channels of an image.

Parameters

device	The device the filter will run on
histogramInfo	Pointer to the MPSHistogramInfo struct

Returns

A valid MPSImageHistogramEqualization object or nil, if failure.

5.89.3 Property Documentation

5.89.3.1 histogramInfo

```
- histogramInfo [read], [nonatomic], [assign]
```

Return a structure describing the histogram content Returns a MPSImageHistogramInfo structure describing the format of the histogram.

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

- MPSImageHistogram.h

5.90 MPSImageHistogramInfo Struct Reference

Specifies information to compute the histogram for channels of an image.

```
#include <MPSImageHistogram.h>
```

Public Attributes

- NSUInteger numberOfHistogramEntries
- BOOL histogramForAlpha
- vector_float4 minPixelValue
- vector_float4 maxPixelValue

5.90.1 Detailed Description

Specifies information to compute the histogram for channels of an image.

MPSImageHistogram.h MetalPerformanceShaders.framework

Copyright

Copyright (c) 2015 Apple Inc. All rights reserved. MetalPerformanceShaders histogram filters

5.90.2 Member Data Documentation

5.90.2.1 histogramForAlpha

```
BOOL MPSImageHistogramInfo::histogramForAlpha
```

Specifies whether the histogram for the alpha channel should be computed or not.

5.90.2.2 maxPixelValue

```
vector_float4 MPSImageHistogramInfo::maxPixelValue
```

Specifies the maximum pixel value. Any pixel value greater than this will be clipped to this value (for the purposes of histogram calculation), and assigned to the first histogram entry. This maximum value is applied to each of the four channels separately.

5.90.2.3 minPixelValue

```
vector_float4 MPSImageHistogramInfo::minPixelValue
```

Specifies the minimum pixel value. Any pixel value less than this will be clipped to this value (for the purposes of histogram calculation), and assigned to the first histogram entry. This minimum value is applied to each of the four channels separately.

5.90.2.4 numberOfHistogramEntries

```
NSUInteger MPSImageHistogramInfo::numberOfHistogramEntries
```

Specifies the number of histogram entries, or "bins" for each channel. For example, if you want 256 histogram bins then numberOfHistogramEntries must be set to 256. The value stored in each histogram bin is a 32-bit unsigned integer. The size of the histogram buffer in which these bins will be stored should be $>=$ numberOfHistogramEntries $*$ sizeof(uint32_t) $*$ number of channels in the image. numberOfHistogramEntries must be a power of 2 and is a minimum of 256 bins.

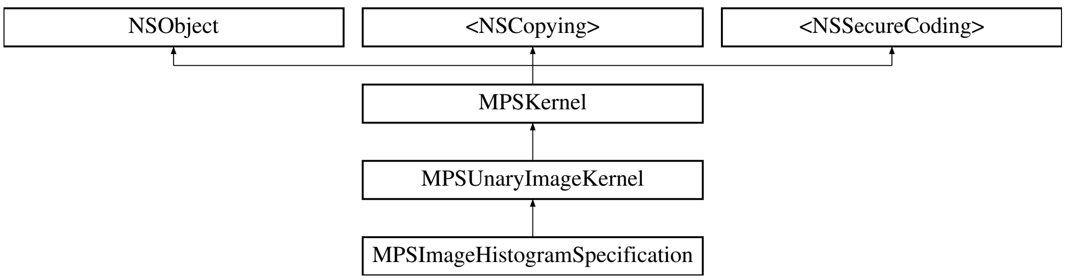
The documentation for this struct was generated from the following file:

- MPSImageHistogram.h

5.91 MPSImageHistogramSpecification Class Reference

```
#import <MPSImageHistogram.h>
```

Inheritance diagram for MPSImageHistogramSpecification:



Instance Methods

- (nonnull instancetype) - initWithDevice:histogramInfo:
- (nullable instancetype) - initWithCoder:device:
- (void) - encodeTransformToCommandBuffer:sourceTexture:sourceHistogram:sourceHistogramOffset↵ :desiredHistogram:desiredHistogramOffset:

Properties

- MPSImageHistogramInfo histogramInfo

Additional Inherited Members

5.91.1 Detailed Description

The MPSImageHistogramSpecification performs a histogram specification operation on an image. It is a generalized version of histogram equalization operation. The histogram specificaiton filter converts the image so that its histogram matches the desired histogram.

5.91.2 Method Documentation

5.91.2.1 encodeTransformToCommandBuffer:sourceTexture:sourceHistogram:sourceHistogramOffset:desiredHistogram:desiredHistogramOffset:()

```
- (void) encodeTransformToCommandBuffer:
            (nonnull id< MTLCommandBuffer >) commandBuffer
            sourceTexture:(nonnull id< MTLTexture >) source
            sourceHistogram:(nonnull id< MTLBuffer >) sourceHistogram
            sourceHistogramOffset:(NSUInteger) sourceHistogramOffset
            desiredHistogram:(nonnull id< MTLBuffer >) desiredHistogram
            desiredHistogramOffset:(NSUInteger) desiredHistogramOffset
```

Encode the transform function to a command buffer using a MTLComputeCommandEncoder. The transform function computes the specification lookup table. The transform function will not begin to execute until after the command buffer has been enqueued and committed. This step will need to be repeated with the new MPSKernel if -copyWithZone:device or -copyWithZone: is called.

Parameters

commandBuffer	A valid MTLCommandBuffer.
source	A valid MTLTexture containing the source image for the filter.
sourceHistogram	A valid MTLBuffer containing the histogram results for the source image. This filter will use these histogram results to generate the cumulative histogram for equalizing the image. The histogram results / channel are stored together. The number of channels for which histogram results are stored is determined by the number of channels in the image. If histogramInfo.histogramForAlpha is false and the source image is RGBA then only histogram results for RGB channels are stored.
sourceHistogramOffset	A byte offset into the sourceHistogram MTLBuffer where the histogram starts. Must conform to alignment requirements for [MTLComputeCommandEncoder setBuffer:offset:atIndex:] offset parameter.
desiredHistogram	A valid MTLBuffer containing the desired histogram results for the source image. The histogram results / channel are stored together. The number of channels for which histogram results are stored is determined by the number of channels in the image. If histogramInfo.histogramForAlpha is false and the source image is RGBA then only histogram results for RGB channels are stored.
desiredHistogramOffset	A byte offset into the desiredHistogram MTLBuffer where the histogram starts. Must conform to alignment requirements for [MTLComputeCommandEncoder setBuffer:offset:atIndex:] offset parameter.

5.91.2.2 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSUnaryImageKernel.

5.91.2.3 initWithDevice:histogramInfo:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            histogramInfo:(const MPSImageHistogramInfo *__nonnull) histogramInfo
```

Specifies information about the histogram for the channels of an image. The MPSImageHistogramSpecification applies a transfor to convert the histogram to a specified histogram. The process is divided into three steps:

1. Call -initWithDevice:histogramInfo: This creates a MPSImageHistogramSpecification object. It is done when the method returns.
2. Call -encodeTransform:sourceTexture:sourceHistogram:sourceHistogramOffset:desiredHistogram: desired↩ HistogramOffset: This creates a privately held image transform which will convert the the distribution of the source histogram to the desired histogram. This process runs on a MTLCommandBuffer when it is committed to a MTLCommandQueue. It must complete before the next step can be run. It may be performed on the same MTLCommandBuffer. The sourceTexture argument is used by encodeTransform to determine the number of channels and therefore which histogram data in sourceHistogram buffer to use. The sourceHistogram and desiredHistogram must have been computed either on the CPU or using the MPSImageHistogram kernel
3. Call -encodeToCommandBuffer:sourceTexture:destinationTexture: to read data from sourceTexture, apply the transform to it and write to destination texture. This step is also done on the GPU on a MTLCommandQueue.

You can reuse the same specification transform on other images to perform the same transform on those images. (Since their starting distribution is probably different, they will probably not arrive at the same distribution as the desired histogram.) This filter usually will not be able to work in place.

Parameters

device	The device the filter will run on
histogramInfo	Pointer to the MPSHistogramInfo struct

Returns

A valid MPSImageHistogramSpecification object or nil, if failure.

5.91.3 Property Documentation

5.91.3.1 histogramInfo

```
- histogramInfo  [read], [nonatomic], [assign]
```

Return a structure describing the histogram content Returns a MPSImageHistogramInfo structure describing the format of the histogram.

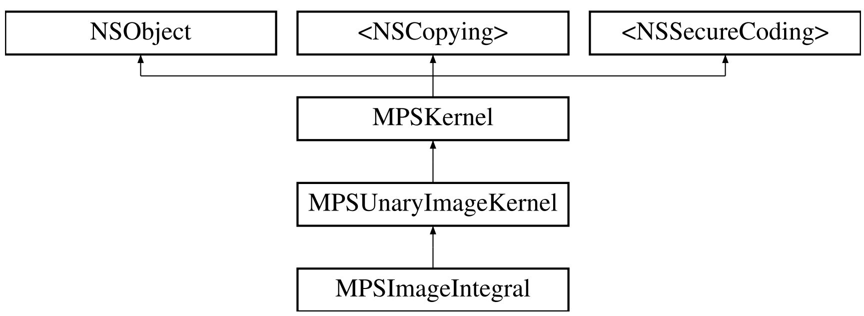
The documentation for this class was generated from the following file:

- MPSImageHistogram.h

5.92 MPSImageIntegral Class Reference

```
#import <MPSImageIntegral.h>
```

Inheritance diagram for MPSImageIntegral:



Additional Inherited Members

5.92.1 Detailed Description

MPSImageIntegral.h MetalPerformanceShaders.framework

Copyright

The MPSImageIntegral calculates the sum of pixels over a specified region in the image. The value at each position is the sum of all pixels in a source image rectangle, sumRect:

```
sumRect.origin = MPSUnaryImageKernel.offset
sumRect.size = dest_position - MPSUnaryImageKernel.clipRect.origin
```

If the channels in the source image are normalized, half-float or floating values, the destination image is recommended to be a 32-bit floating-point image. If the channels in the source image are integer values, it is recommended that an appropriate 32-bit integer image destination format is used.

This kernel accepts uint and int textures in addition to unorm and floating-point textures.

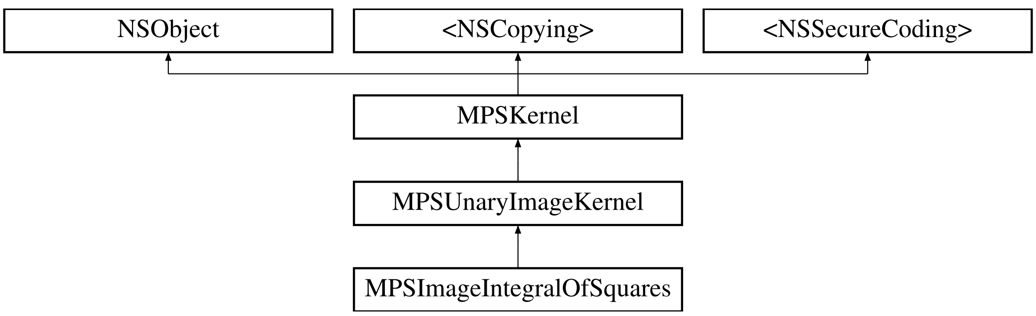
The documentation for this class was generated from the following file:

- MPSImageIntegral.h

5.93 MPSImageIntegralOfSquares Class Reference

```
#import <MPSImageIntegral.h>
```

Inheritance diagram for MPSImageIntegralOfSquares:



Additional Inherited Members

5.93.1 Detailed Description

The MPSImageIntegralOfSquares calculates the sum of squared pixels over a specified region in the image. The value at each position is the sum of all squared pixels in a source image rectangle, sumRect:

```
sumRect.origin = MPSUnaryImageKernel.offset
sumRect.size = dest_position - MPSUnaryImageKernel.clipRect.origin
```

If the channels in the source image are normalized, half-float or floating values, the destination image is recommended to be a 32-bit floating-point image. If the channels in the source image are integer values, it is recommended that an appropriate 32-bit integer image destination format is used.

This kernel accepts uint and int textures in addition to unorm and floating-point textures.

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

- MPSImageIntegral.h

5.94 MPSImageKeypointData Struct Reference

Specifies keypoint information.

`#include <MPSImageKeypoint.h>`

Public Attributes

- vector_ushort2 keypointCoordinate
- float keypointColorValue

5.94.1 Detailed Description

Specifies keypoint information.

5.94.2 Member Data Documentation

5.94.2.1 keypointColorValue

`float MPSImageKeypointData::keypointColorValue`

keypoint color value

5.94.2.2 keypointCoordinate

```
vector_ushort2 MPSImageKeypointData::keypointCoordinate
```

keypoint (x, y) coordinate

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

- MPSImageKeypoint.h

5.95 MPSImageKeypointRangeInfo Struct Reference

Specifies information to find the keypoints in an image.

```
#include <MPSImageKeypoint.h>
```

Public Attributes

- NSUInteger maximumKeypoints
- float minimumThresholdValue

5.95.1 Detailed Description

Specifies information to find the keypoints in an image.

MPSImageKeypoint.h MetalPerformanceShaders.framework

Copyright

5.95.2 Member Data Documentation

5.95.2.1 maximumKeypoints

```
NSUInteger MPSImageKeypointRangeInfo::maximumKeypoints
```

maximum number of keypoints

5.95.2.2 minimumThresholdValue

```
float MPSImageKeypointRangeInfo::minimumThresholdValue
```

minimum threshold value - value between 0.0 and 1.0f

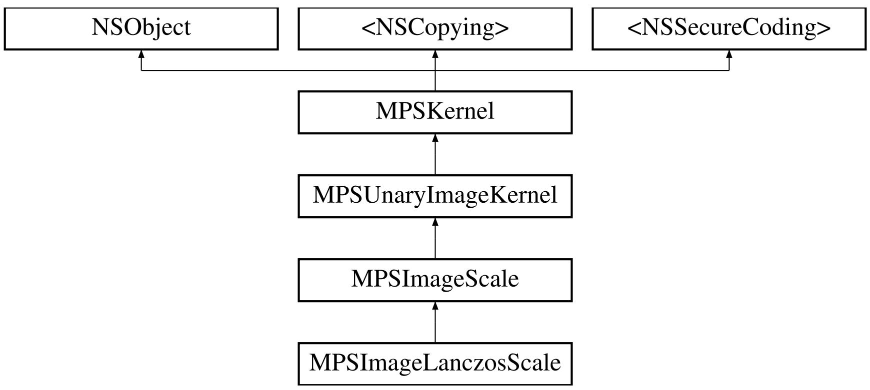
The documentation for this struct was generated from the following file:

- MPSImageKeypoint.h

5.96 MPSImageLanczosScale Class Reference

```
#import <MPSImageResampling.h>
```

Inheritance diagram for MPSImageLanczosScale:



Instance Methods

- (nonnull instancetype) - initWithDevice:
- (nullable instancetype) - initWithCoder:device:

Additional Inherited Members

5.96.1 Detailed Description

Resize an image and / or change its aspect ratio The MPSImageLanczosScale filter can be used to resample an existing image using a different sampling frequency in each dimension. This can be used to enlarge or reduce the size of an image, or change the aspect ratio of an image. The filter uses a Lanczos resampling algorithm which typically produces better quality for photographs, but is slower than linear sampling using the GPU texture units. Lanczos downsampling does not require a low pass filter to be applied before it is used. Because the resampling function has negative lobes, Lanczos can result in ringing near sharp edges, making it less suitable for vector art.

5.96.2 Method Documentation

5.96.2.1 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSImageScale.

5.96.2.2 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

a pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSImageScale.

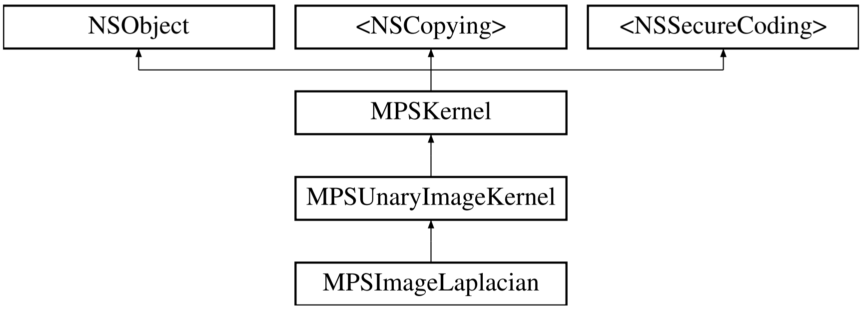
The documentation for this class was generated from the following file:

- MPSImageResampling.h

5.97 MPSImageLaplacian Class Reference

```
#import <MPSImageConvolution.h>
```

Inheritance diagram for MPSImageLaplacian:



Properties

- float bias

Additional Inherited Members

5.97.1 Detailed Description

The MPSImageLaplacian is an optimized variant of the MPSImageConvolution filter provided primarily for ease of use. This filter uses an optimized convolution filter with a 3 x 3 kernel with the following weights: [0 1 0 1 -4 1 0 1 0]

The optimized convolution filter used by MPSImageLaplacian can also be used by creating a MPSImageConvolution object with kernelWidth = 3, kernelHeight = 3 and weights as specified above.

5.97.2 Property Documentation

5.97.2.1 bias

```
- bias  [read], [write], [nonatomic], [assign]
```

The bias is a value to be added to convolved pixel before it is converted back to the storage format. It can be used to convert negative values into a representable range for a unsigned MTLPixelFormat. For example, many edge detection filters produce results in the range [-k,k]. By scaling the filter weights by 0.5/k and adding 0.5, the results will be in range [0,1] suitable for use with unorm formats. It can be used in combination with renormalization of the filter weights to do video ranging as part of the convolution effect. It can also just be used to increase the brightness of the image.

Default value is 0.0f.

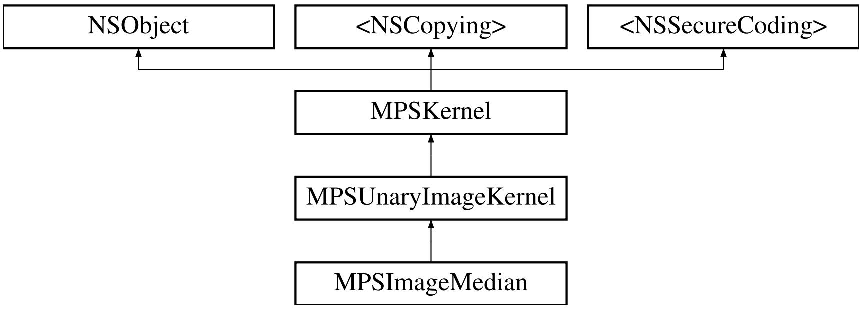
The documentation for this class was generated from the following file:

- MPSImageConvolution.h

5.98 MPSImageMedian Class Reference

`#import <MPSImageMedian.h>`

Inheritance diagram for MPSImageMedian:



Instance Methods

- (nonnull instancetype) - initWithDevice:kernelDiameter:
- (nullable instancetype) - initWithCoder:device:
- (nonnull instancetype) - initWithDevice:

Class Methods

- (NSUInteger) + maxKernelDiameter
- (NSUInteger) + minKernelDiameter

Properties

- NSUInteger kernelDiameter

Additional Inherited Members

5.98.1 Detailed Description

MPSImageMedian.h MetalPerformanceShaders.framework

Copyright

 MetalPerformanceShaders median filters

The MPSImageMedian applies a median filter to an image. A median filter finds the median color value for each channel within a kernelDiameter x kernelDiameter window surrounding the pixel of interest. It is a common means of noise reduction and also as a smoothing filter with edge preserving qualities.

NOTE: The MPSImageMedian filter currently only supports images with $<=$ 8 bits/channel.

5.98.2 Method Documentation

5.98.2.1 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSUnaryImageKernel.

5.98.2.2 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

a pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSUnaryImageKernel.

5.98.2.3 initWithDevice:kernelDiameter:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            kernelDiameter:(NSUInteger) kernelDiameter
```

Initialize a filter for a particular kernel size and device

Parameters

device	The device the filter will run on
kernelDiameter	Diameter of the median filter. Must be an odd number.

Returns

A valid object or nil, if failure.

5.98.2.4 maxKernelDiameter()

```
+ (NSUInteger) maxKernelDiameter
```

The maximum diameter in pixels of the filter window supported by the median filter.

5.98.2.5 minKernelDiameter()

```
+ (NSUInteger) minKernelDiameter
```

The minimum diameter in pixels of the filter window supported by the median filter.

5.98.3 Property Documentation

5.98.3.1 kernelDiameter

```
- kernelDiameter  [read], [nonatomic], [assign]
```

The diameter in pixels of the filter window. The median filter is applied to a kernelDiameter x kernelDiameter window of pixels centered on the corresponding source pixel for each destination pixel. The kernel diameter must be an odd number.

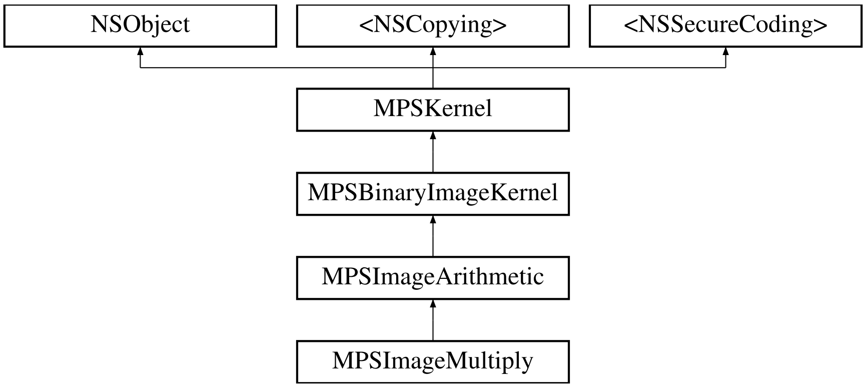
The documentation for this class was generated from the following file:

- MPSImageMedian.h

5.99 MPSImageMultiply Class Reference

```
#import <MPSImageMath.h>
```

Inheritance diagram for MPSImageMultiply:



Instance Methods

- (nonnull instancetype) - initWithDevice:

Additional Inherited Members

5.99.1 Detailed Description

This depends on Metal.framework. Specifies the multiplication operator. For each pixel in the primary source image (x) and each pixel in a secondary source image (y), it applies the following function: result = ((primaryScale $*$ x) $*$ (secondaryScale $*$ y)) + bias.

5.99.2 Method Documentation

5.99.2.1 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Initialize the multiplication operator

Parameters

device	The device the filter will run on.

Returns

A valid MPSImageMultiply object or nil, if failure.

Reimplemented from MPSImageArithmetic.

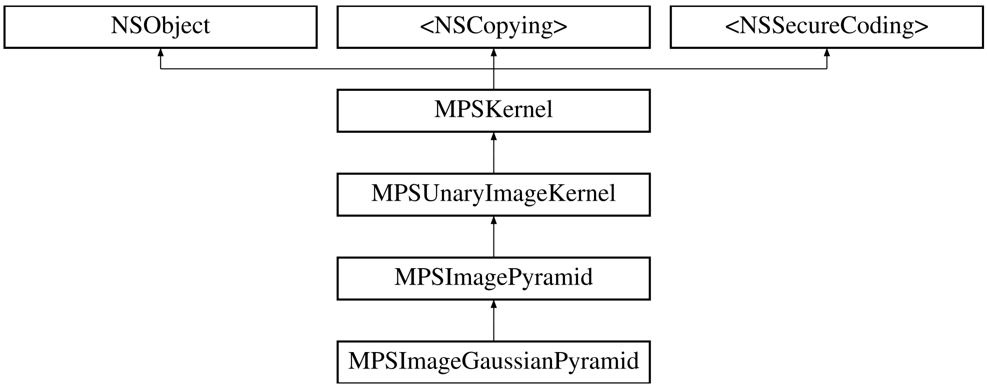
The documentation for this class was generated from the following file:

- MPSImageMath.h

5.100 MPSImagePyramid Class Reference

```
#import <MPSImageConvolution.h>
```

Inheritance diagram for MPSImagePyramid:



Instance Methods

- (nonnull instancetype) - initWithDevice:
- (nonnull instancetype) - initWithDevice:centerWeight:
- (nonnull instancetype) - initWithDevice:kernelWidth:kernelHeight:weights:
- (nullable instancetype) - initWithCoder:device:

Properties

- NSUInteger kernelHeight
- NSUInteger kernelWidth

Additional Inherited Members

5.100.1 Detailed Description

The MPSImagePyramid is a base class for creating different kinds of pyramid images

```
Currently supported pyramid-types are:
@ref MPSImageGaussianPyramid

The Gaussian image pyramid kernel is enqueued as a in-place operation using
@ref MPSUnaryImageKernel::encodeToCommandBuffer:inPlaceTexture:fallbackCopyAllocator:
and all mipmap levels after level=1, present in the provided image are filled using
the provided filtering kernel. The fallbackCopyAllocator parameter is not used.

The Gaussian image pyramid filter ignores @ref clipRect and @ref offset and fills
the entire mipmap levels.
```

Note

Make sure your texture type is compatible with mipmapping and supports texture views (see MTLTexture↩UsagePixelFormatView).
Recall the size of the nth mipmap level:

```
w_n = max(1, floor(w_0 / 2^n))
h_n = max(1, floor(h_0 / 2^n)),
```

where w_0, h_0 are the zeroth level width and height. ie the image dimensions themselves.

5.100.2 Method Documentation

5.100.2.1 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability See MPSKernel::initWithCoder.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSCNNPooling
device	The MTLDevice on which to make the MPSCNNPooling

Returns

A new MPSCNNPooling object, or nil if failure.

Reimplemented from MPSUnaryImageKernel.

5.100.2.2 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Initialize a downwards 5-tap image pyramid with the default filter kernel and device

Parameters

device	The device the filter will run on

The filter kernel is the outer product of w = [1/16, 1/4, 3/8, 1/4, 1/16]^T, with itself

Returns

A valid object or nil, if failure.

Reimplemented from MPSUnaryImageKernel.

5.100.2.3 initWithDevice:centerWeight:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            centerWeight:(float) centerWeight
```

Initialize a downwards 5-tap image pyramid with a central weight parameter and device

Parameters

device	The device the filter will run on
centerWeight	Defines form of the filter-kernel through the outer product ww^T, where w = [(1/4 - a/2), 1/4, a, 1/4, (1/4 - a/2)]^T and 'a' is centerWeight.

Returns

A valid object or nil, if failure.

5.100.2.4 initWithDevice:kernelWidth:kernelHeight:weights:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            kernelWidth:(NSUInteger) kernelWidth
            kernelHeight:(NSUInteger) kernelHeight
            weights:(const float *__nonnull) kernelWeights
```

Initialize a downwards n-tap pyramid with a custom filter kernel and device

Parameters

device	The device the filter will run on
kernelWidth	The width of the filtering kernel. See MPSImageConvolution.
kernelHeight	The height of the filtering kernel. See MPSImageConvolution.
kernelWeights	A pointer to an array of kernelWidth $*$ kernelHeight values to be used as the kernel. These are in row major order. See MPSImageConvolution.

Returns

A valid object or nil, if failure.

5.100.3 Property Documentation

5.100.3.1 kernelHeight

```
- kernelHeight  [read], [nonatomic], [assign]
```

The height of the filter window. Must be an odd number.

5.100.3.2 kernelWidth

```
- kernelWidth  [read], [nonatomic], [assign]
```

The width of the filter window. Must be an odd number.

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

- MPSImageConvolution.h

5.101 MPSImageReadWriteParams Struct Reference

```
#include <MPSImage.h>
```

Public Attributes

- NSUInteger featureChannelOffset
- NSUInteger numberOfFeatureChannelsToReadWrite

5.101.1 Detailed Description

these parameters are passed in to allow user to read/write to a particular set of featureChannels in an MPSImage

5.101.2 Member Data Documentation

5.101.2.1 featureChannelOffset

```
NSUInteger MPSImageReadWriteParams::featureChannelOffset
```

featureChannel offset from which to read/write featureChannels, this should be a multiple of 4

5.101.2.2 numberOfFeatureChannelsToReadWrite

```
NSUInteger MPSImageReadWriteParams::numberOfFeatureChannelsToReadWrite
```

is number of featureChannels, should be greater than 0 and multiple of 4 unless featureChannelOffset is 0

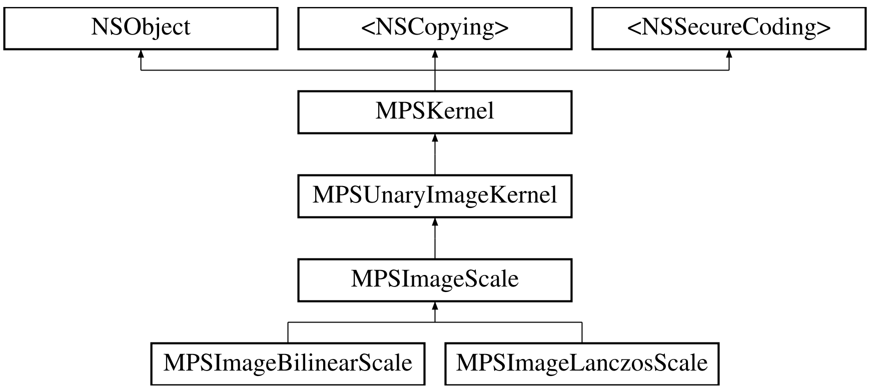
The documentation for this struct was generated from the following file:

- MPSCore.framework/Headers/MPSImage.h

5.102 MPSImageScale Class Reference

```
#import <MPSImageResampling.h>
```

Inheritance diagram for MPSImageScale:



Instance Methods

- (nonnull instancetype) - initWithDevice:
- (nullable instancetype) - initWithCoder:device:

Properties

- const MPSScaleTransform ∗ scaleTransform

Additional Inherited Members

5.102.1 Detailed Description

MPSImageResampling.h MetalPerformanceShaders

Copyright

 Resampling filters for MetalPerformanceShaders

Resize an image and / or change its aspect ratio The MPSImageScale filter can be used to resample an existing image using a different sampling frequency in each dimension. This can be used to enlarge or reduce the size of an image, or change the aspect ratio of an image.

The resample methods supported are: Bilinear Bicubcic Lanczos

5.102.2 Method Documentation

5.102.2.1 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSUnaryImageKernel.

Reimplemented in MPSImageLanczosScale, and MPSImageBilinearScale.

5.102.2.2 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

a pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSUnaryImageKernel.

Reimplemented in MPSImageLanczosScale, and MPSImageBilinearScale.

5.102.3 Property Documentation

5.102.3.1 scaleTransform

```
- scaleTransform  [read], [write], [nonatomic], [assign]
```

An optional transform that describes how to scale and translate the source image If the scaleTransform is NU↩ LL, then the MPSImageLanczosScale filter will rescale the image so that the source image fits exactly into the destination texture. If the transform is not NULL, then the transform is used for determining how to map the source image to the destination. Default: NULL

When the scaleTransform is set to non-NULL, the values pointed to by the new scaleTransform are copied to object storage, and the pointer is updated to point to internal storage. Do not attempt to free it. You may free your copy of the MPSScaleTransform as soon as the property set operation is complete.

When calculating a scaleTransform, use the limits of the bounding box for the intended source region of interest and the destination clipRect. Adjustments for pixel center coordinates are handled internally to the function. For example, the scale transform to convert the entire source image to the entire destination image size (clipRect = MPSRectNoClip) would be:

```
scaleTransform.scaleX = (double) dest.width / source.width;
scaleTransform.scaleY = (double) dest.height / source.height;
scaleTransform.translateX = scaleTransform.
      translateY = 0.0;
```

The translation parameters allow you to adjust the region of the source image used to create the destination image. They are in destination coordinates. To place the top left corner of the destination clipRect to represent the position {x,y} in source coordinates, we solve for the translation based on the standard scale matrix operation for each axis:

```
dest_position = source_position * scale + translation;
translation = dest_position - source_position * scale;
```

For the top left corner of the clipRect, the dest_position is considered to be {0,0}. This gives us a translation of:

```
scaleTransform.translateX = -source_origin.x *
      scaleTransform.scaleX;
scaleTransform.translateY = -source_origin.y *
      scaleTransform.scaleY;
```

One would typically use non-zero translations to do tiling, or provide a resized view into a internal segment of an image.

Changing the Lanczos scale factor may trigger recalculation of signficant state internal to the object when the filter is encoded to the command buffer. The scale factor is scaleTransform->scaleX,Y, or the ratio of source and destination image sizes if scaleTransform is NULL. Reuse a MPSImageLanczosScale object for frequently used scalings to avoid redundantly recreating expensive resampling state.

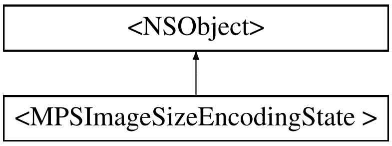
The documentation for this class was generated from the following file:

- MPSImageResampling.h

5.103 <MPSImageSizeEncodingState > Protocol Reference

```
#import <MPSNeuralNetworkTypes.h>
```

Inheritance diagram for <MPSImageSizeEncodingState >:



Properties

- NSUInteger sourceWidth
- NSUInteger sourceHeight

5.103.1 Detailed Description

MPSStates conforming to this protocol contain information about a image size elsewhere in the graph In some graphs a sequence of operations are done, then they are undone ins a series of 'reverse' operations. Examples might be pooling vs unpooling / upsampling, or convolution vs. convolution transpose. In such cases, the 'reverse' pass generally is converting from a smaller image to a larger image, and there is insufficient information to do this correctly. Several answers exist and we don't know which is correct.

As an example, consider trying to 'undo' integer division with a multiplication. The expression c = a/b is incomplete because there is also a remainder, which may constitute information lost. If we want to reconstitute a based on c and b, we need to use a = c $*$ b + remainder, not just a = c$*$b. Similarly, when undoing a downsizing operation, we need the original size to find which answer in the range of a = c$*$b + [0,b-1] is the right one.

5.103.2 Property Documentation

5.103.2.1 sourceHeight

```
- (NSUInteger MPSImageSizeEncodingState) sourceHeight  [read], [nonatomic], [assign]
```

The height of the source image passed to MPSCNNConvolution encode call.

5.103.2.2 sourceWidth

```
- (NSUInteger MPSImageSizeEncodingState) sourceWidth  [read], [nonatomic], [assign]
```

The width of the source image passed to MPSCNNConvolution encode call.

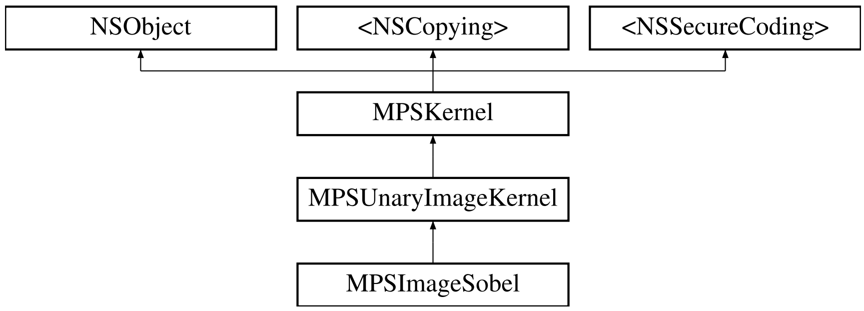
The documentation for this protocol was generated from the following file:

- MPSNeuralNetworkTypes.h

5.104 MPSImageSobel Class Reference

```
#import <MPSImageConvolution.h>
```

Inheritance diagram for MPSImageSobel:



Instance Methods

- (nonnull instancetype) - initWithDevice:
- (nonnull instancetype) - initWithDevice:linearGrayColorTransform:
- (nullable instancetype) - initWithCoder:device:

Properties

- const float $*$ colorTransform

Additional Inherited Members

5.104.1 Detailed Description

The MPSImageSobel implements the Sobel filter. When the color model (e.g. RGB, two-channel, grayscale, etc.) of source and destination textures match, the filter is applied to each channel separately. If the destination is monochrome (single channel) but source multichannel, the pixel values are converted to grayscale before applying Sobel operator using the linear gray color transform vector (v).

```
Luminance = v[0] * pixel.x + v[1] * pixel.y + v[2] * pixel.z;
```

5.104.2 Method Documentation

5.104.2.1 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSUnaryImageKernel.

5.104.2.2 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Initialize a Sobel filter on a given device using the default color transform. Default: BT.601/JPEG {0.299f, 0.587f, 0.114f}

For non-default conversion matrices, use -initWithDevice:linearGrayColorTransform:

Parameters

device	The device the filter will run on

Returns

A valid object or nil, if failure.

Reimplemented from MPSUnaryImageKernel.

5.104.2.3 initWithDevice:linearGrayColorTransform:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            linearGrayColorTransform:(const float *__nonnull) transform
```

Initialize a Sobel filter on a given device with a non-default color transform

Parameters

device	The device the filter will run on
transform	Array of three floats describing the rgb to gray scale color transform. `Luminance = transform[0] * pixel.x + transform[1] * pixel.y + transform[2] * pixel.z;`

Returns

A valid object or nil, if failure.

5.104.3 Property Documentation

5.104.3.1 colorTransform

```
- colorTransform  [read], [nonatomic], [assign]
```

Returns a pointer to the array of three floats used to convert RGBA, RGB or RG images to the destination format when the destination is monochrome.

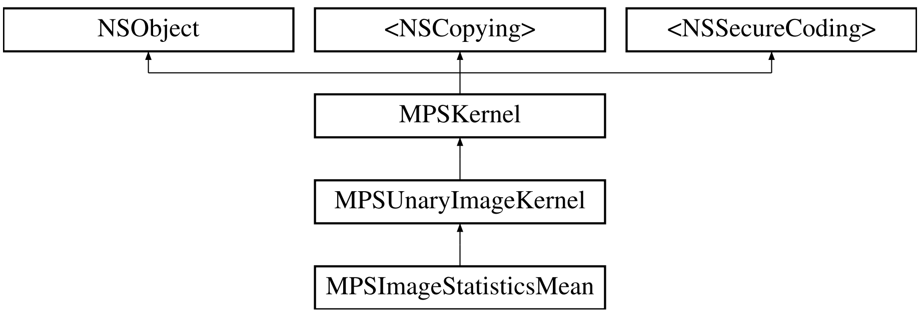
The documentation for this class was generated from the following file:

- MPSImageConvolution.h

5.105 MPSImageStatisticsMean Class Reference

```
#import <MPSImageStatistics.h>
```

Inheritance diagram for MPSImageStatisticsMean:



Instance Methods

- (nonnull instancetype) - initWithDevice:
- (nullable instancetype) - initWithCoder:device:

Properties

- MTLRegion clipRectSource

Additional Inherited Members

5.105.1 Detailed Description

The MPSImageStatisticsMean computes the mean for a given region of an image.

5.105.2 Method Documentation

5.105.2.1 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSUnaryImageKernel.

5.105.2.2 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Specifies information to apply the statistics mean operation on an image.

Parameters

device	The device the filter will run on

Returns

A valid MPSImageStatisticsMean object or nil, if failure.

Reimplemented from MPSUnaryImageKernel.

5.105.3 Property Documentation

5.105.3.1 clipRectSource

```
- clipRectSource  [read], [write], [nonatomic], [assign]
```

The source rectangle to use when reading data. A MTLRegion that indicates which part of the source to read. If the clipRectSource does not lie completely within the source image, the intersection of the image bounds and clip←RectSource will be used. The clipRectSource replaces the MPSUnaryImageKernel offset parameter for this filter. The latter is ignored. Default: MPSRectNoClip, use the entire source texture.

The clipRect specified in MPSUnaryImageKernel is used to control the origin in the destination texture where the mean value is written.

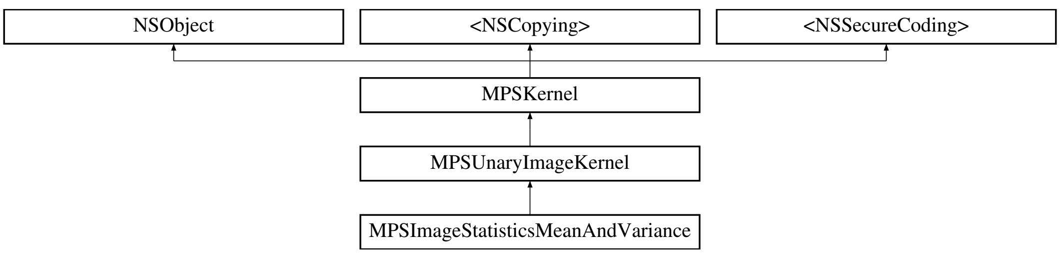
The documentation for this class was generated from the following file:

- MPSImageStatistics.h

5.106 MPSImageStatisticsMeanAndVariance Class Reference

```
#import <MPSImageStatistics.h>
```

Inheritance diagram for MPSImageStatisticsMeanAndVariance:



Instance Methods

- (nonnull instancetype) - initWithDevice:
- (nullable instancetype) - initWithCoder:device:

Properties

- MTLRegion clipRectSource

Additional Inherited Members

5.106.1 Detailed Description

The MPSImageStatisticsMeanAndVariance computes the mean and variance for a given region of an image. The mean and variance values are written to the destination image at the following pixel locations:

- mean value is written at pixel location (0, 0)
- variance value is written at pixel location (1, 0)

5.106.2 Method Documentation

5.106.2.1 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSUnaryImageKernel.

5.106.2.2 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Specifies information to apply the statistics mean operation on an image.

Parameters

device	The device the filter will run on

Returns

A valid MPSImageStatisticsMeanAndVariance object or nil, if failure.

Reimplemented from MPSUnaryImageKernel.

5.106.3 Property Documentation

5.106.3.1 clipRectSource

```
- clipRectSource  [read], [write], [nonatomic], [assign]
```

The source rectangle to use when reading data. A MTLRegion that indicates which part of the source to read. If the clipRectSource does not lie completely within the source image, the intersection of the image bounds and clipRectSource will be used. The clipRectSource replaces the MPSUnaryImageKernel offset parameter for this filter. The latter is ignored. Default: MPSRectNoClip, use the entire source texture.

The clipRect specified in MPSUnaryImageKernel is used to control the origin in the destination texture where the mean value is written.

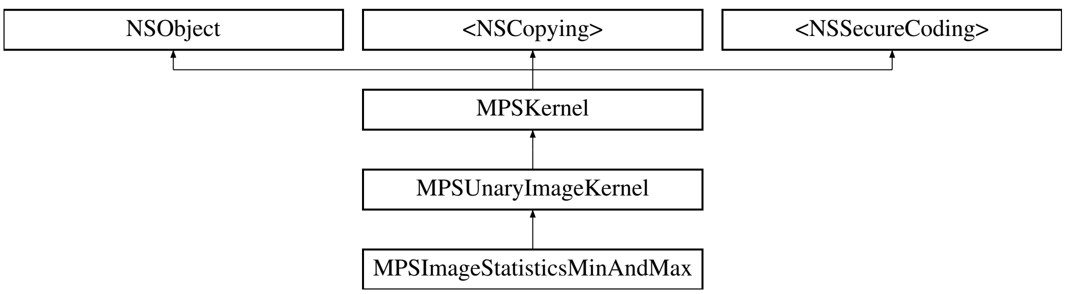
The documentation for this class was generated from the following file:

- MPSImageStatistics.h

5.107 MPSImageStatisticsMinAndMax Class Reference

```
#import <MPSImageStatistics.h>
```

Inheritance diagram for MPSImageStatisticsMinAndMax:



Instance Methods

- (nonnull instancetype) - initWithDevice:
- (nullable instancetype) - initWithCoder:device:

Properties

- MTLRegion clipRectSource

Additional Inherited Members

5.107.1 Detailed Description

MPSImageStatistics.h MetalPerformanceShaders.framework

Copyright

 MetalPerformanceShaders image statistics filters

The MPSImageStatisticsMinAndMax computes the minimum and maximum pixel values for a given region of an image. The min and max values are written to the destination image at the following pixel locations:

- min value is written at pixel location (0, 0)
- max value is written at pixel location (1, 0)

5.107.2 Method Documentation

5.107.2.1 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSUnaryImageKernel.

5.107.2.2 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Specifies information to apply the statistics min-max operation on an image.

Parameters

device	The device the filter will run on

Returns

A valid MPSImageStatisticsMinAndMax object or nil, if failure.

Reimplemented from MPSUnaryImageKernel.

5.107.3 Property Documentation

5.107.3.1 clipRectSource

```
- clipRectSource  [read], [write], [nonatomic], [assign]
```

The source rectangle to use when reading data. A MTLRegion that indicates which part of the source to read. If the clipRectSource does not lie completely within the source image, the intersection of the image bounds and clip↵ RectSource will be used. The clipRectSource replaces the MPSUnaryImageKernel offset parameter for this filter. The latter is ignored. Default: MPSRectNoClip, use the entire source texture.

The clipRect specified in MPSUnaryImageKernel is used to control the origin in the destination texture where the min, max values are written. The clipRect.width must be >=2. The clipRect.height must be >= 1.

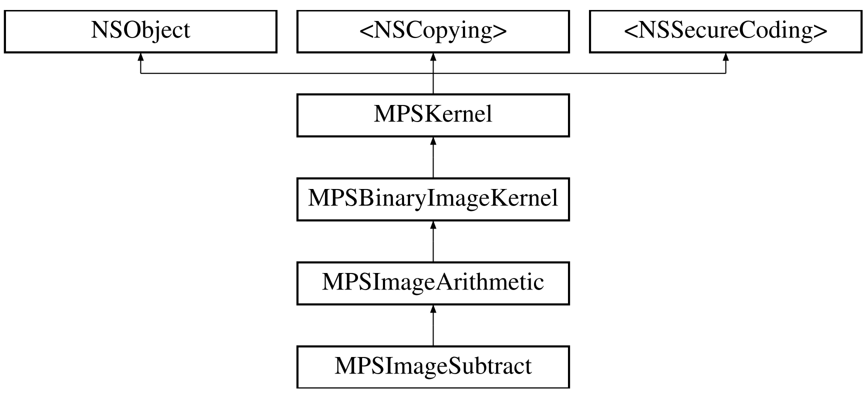
The documentation for this class was generated from the following file:

- MPSImageStatistics.h

5.108 MPSImageSubtract Class Reference

```
#import <MPSImageMath.h>
```

Inheritance diagram for MPSImageSubtract:



Instance Methods

- (nonnull instancetype) - initWithDevice:

Additional Inherited Members

5.108.1 Detailed Description

This depends on Metal.framework. Specifies the subtraction operator. For each pixel in the primary source image (x) and each pixel in a secondary source image (y), it applies the following function: result = ((primaryScale $*$ x) - (secondaryScale $*$ y)) + bias.

5.108.2 Method Documentation

5.108.2.1 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Initialize the subtraction operator

Parameters

device	The device the filter will run on.

Returns

A valid MPSImageSubtract object or nil, if failure.

Reimplemented from MPSImageArithmetic.

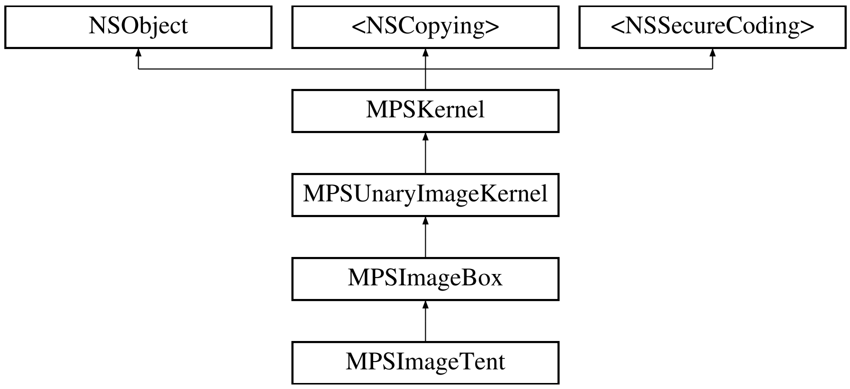
The documentation for this class was generated from the following file:

- MPSImageMath.h

5.109 MPSImageTent Class Reference

```
#import <MPSImageConvolution.h>
```

Inheritance diagram for MPSImageTent:



Additional Inherited Members

5.109.1 Detailed Description

The box filter, while fast, may yield square-ish looking blur effects. However, multiple passes of the box filter tend to smooth out with each additional pass. For example, two 3-wide box blurs produces the same effective convolution as a 5-wide tent blur:

```
1   1   1
    1   1   1
+       1   1   1
================
1   2   3   2   1
```

Addition passes tend to approximate a gaussian line shape.

The MPSImageTent convolves an image with a tent filter. These form a tent shape with incrementally increasing sides, for example:

```
1   2   3   2   1
```

```
1   2   1
2   4   2
1   2   1
```

Like the box filter, this arrangement allows for much faster algorithms, espcially for for larger blur radii but with a more pleasing appearance.

The tent blur is a separable filter. The implementation is aware of this and will act accordingly to give best performance for multi-dimensional blurs.

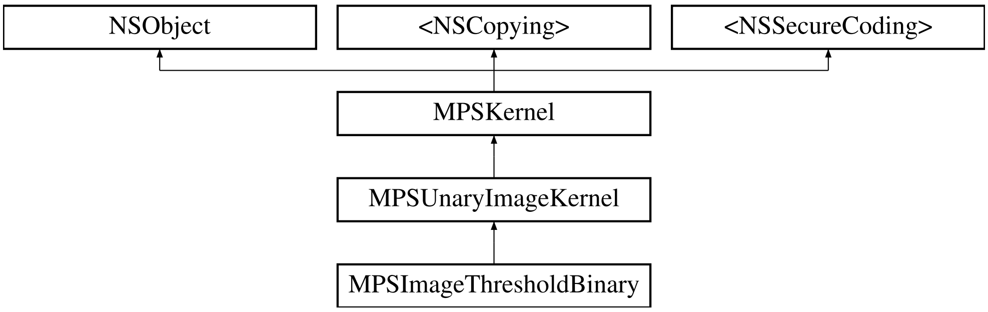
The documentation for this class was generated from the following file:

- MPSImageConvolution.h

5.110 MPSImageThresholdBinary Class Reference

```
#import <MPSImageThreshold.h>
```

Inheritance diagram for MPSImageThresholdBinary:



Instance Methods

- (nonnull instancetype) - initWithDevice:thresholdValue:maximumValue:linearGrayColorTransform:
- (nullable instancetype) - initWithCoder:device:
- (nonnull instancetype) - initWithDevice:

Properties

- float thresholdValue
- float maximumValue
- const float $*$ transform

Additional Inherited Members

5.110.1 Detailed Description

MPSImageThreshold.h MetalPerformanceShaders

Copyright

Copyright (c) 2015 Apple Inc. All rights reserved. MetalPerformanceShaders thresholding filters

The MPSThreshold filter applies a fixed-level threshold to each pixel in the image. The threshold functions convert a single channel image to a binary image. If the input image is not a single channel image, convert the inputimage to a single channel luminance image using the linearGrayColorTransform and then apply the threshold. The Threshold↩ Binary function is: destinationPixelValue = sourcePixelValue $>$ thresholdValue ? maximumValue : 0

5.110.2 Method Documentation

5.110.2.1 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSUnaryImageKernel.

5.110.2.2 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

a pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSUnaryImageKernel.

5.110.2.3 initWithDevice:thresholdValue:maximumValue:linearGrayColorTransform:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            thresholdValue:(float) thresholdValue
            maximumValue:(float) maximumValue
            linearGrayColorTransform:(const float *__nullable) transform
```

initialize a MPSImageThresholdBinary filter

Parameters

device	The device the filter will run on
thresholdValue	The threshold value to use
maximumValue	The maximum value to use
transform	This matrix is an array of 3 floats. The default if no transform is specifed is BT.601/JPEG: {0.299f, 0.587f, 0.114f};

5.110.3 Property Documentation

5.110.3.1 maximumValue

```
- maximumValue [read], [nonatomic], [assign]
```

The maximum value used to init the threshold filter

5.110.3.2 thresholdValue

```
- thresholdValue [read], [nonatomic], [assign]
```

The threshold value used to init the threshold filter

5.110.3.3 transform

```
- transform [read], [nonatomic], [assign]
```

The color transform used to init the threshold filter

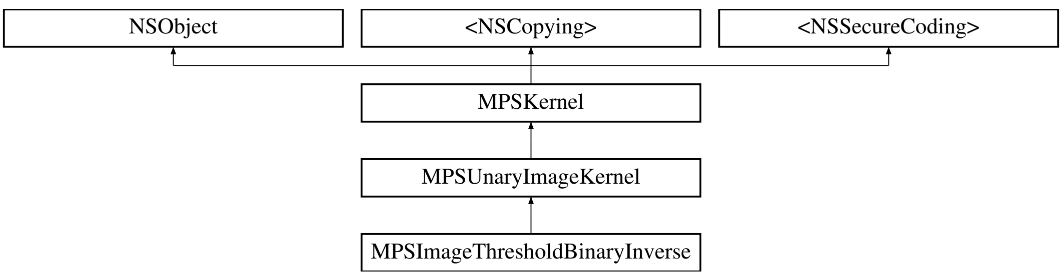
The documentation for this class was generated from the following file:

- MPSImageThreshold.h

5.111 MPSImageThresholdBinaryInverse Class Reference

```
#import <MPSImageThreshold.h>
```

Inheritance diagram for MPSImageThresholdBinaryInverse:



Instance Methods

- (nonnull instancetype) - initWithDevice:thresholdValue:maximumValue:linearGrayColorTransform:
- (nullable instancetype) - initWithCoder:device:
- (nonnull instancetype) - initWithDevice:

Properties

- float thresholdValue
- float maximumValue
- const float ∗ transform

Additional Inherited Members

5.111.1 Detailed Description

The MPSImageThresholdBinaryInverse filter applies a fixed-level threshold to each pixel in the image. The threshold functions convert a single channel image to a binary image. If the input image is not a single channel image, convert the inputimage to a single channel luminance image using the linearGrayColorTransform and then apply the threshold. The ThresholdBinaryInverse function is: destinationPixelValue = sourcePixelValue > thresholdValue ? 0 : maximumValue

5.111.2 Method Documentation

5.111.2.1 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSUnaryImageKernel.

5.111.2.2 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

a pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSUnaryImageKernel.

5.111.2.3 initWithDevice:thresholdValue:maximumValue:linearGrayColorTransform:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            thresholdValue:(float) thresholdValue
            maximumValue:(float) maximumValue
            linearGrayColorTransform:(const float *__nullable) transform
```

initialize a MPSImageThresholdBinaryInverse filter

Parameters

device	The device the filter will run on
thresholdValue	The threshold value to use
maximumValue	The maximum value to use
transform	This matrix is an array of 3 floats. The default if no transform is specifed is BT.601/JPEG: {0.299f, 0.587f, 0.114f};

5.111.3 Property Documentation

5.111.3.1 maximumValue

```
- maximumValue  [read], [nonatomic], [assign]
```

The maximum value used to init the threshold filter

5.111.3.2 thresholdValue

```
- thresholdValue  [read], [nonatomic], [assign]
```

The threshold value used to init the threshold filter

5.111.3.3 transform

```
- transform [read], [nonatomic], [assign]
```

The color transform used to init the threshold filter

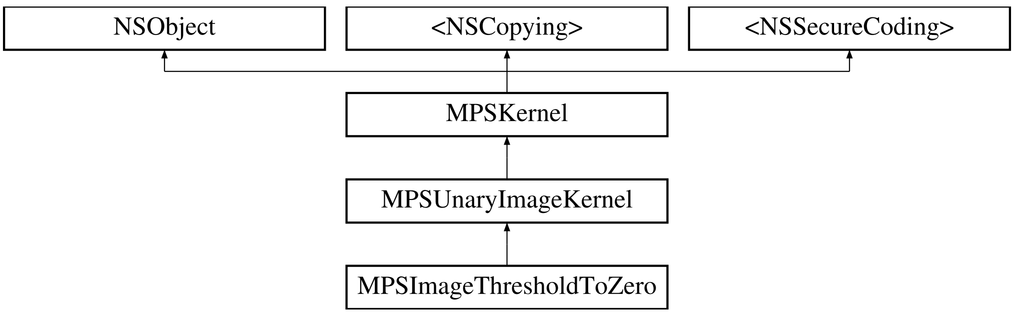
The documentation for this class was generated from the following file:

- MPSImageThreshold.h

5.112 MPSImageThresholdToZero Class Reference

```
#import <MPSImageThreshold.h>
```

Inheritance diagram for MPSImageThresholdToZero:



Instance Methods

- (nonnull instancetype) - initWithDevice:thresholdValue:linearGrayColorTransform:
- (nonnull instancetype) - initWithDevice:
- (nullable instancetype) - initWithCoder:device:

Properties

- float thresholdValue
- const float $*$ transform

Additional Inherited Members

5.112.1 Detailed Description

The MPSImageThresholdToZero filter applies a fixed-level threshold to each pixel in the image. The threshold functions convert a single channel image to a binary image. If the input image is not a single channel image, convert the inputimage to a single channel luminance image using the linearGrayColorTransform and then apply the threshold. The ThresholdToZero function is: destinationPixelValue = sourcePixelValue $>$ thresholdValue ? sourcePixelValue : 0

5.112.2 Method Documentation

5.112.2.1 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSUnaryImageKernel.

5.112.2.2 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

a pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSUnaryImageKernel.

5.112.2.3 initWithDevice:thresholdValue:linearGrayColorTransform:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            thresholdValue:(float) thresholdValue
            linearGrayColorTransform:(const float *__nullable) transform
```

initialize a MPSImageThresholdToZero filter

Parameters

device	The device the filter will run on
thresholdValue	The threshold value to use
transform	This matrix is an array of 3 floats. The default if no transform is specifed is BT.601/JPEG: {0.299f, 0.587f, 0.114f};

5.112.3 Property Documentation

5.112.3.1 thresholdValue

```
- thresholdValue [read], [nonatomic], [assign]
```

The threshold value used to init the threshold filter

5.112.3.2 transform

```
- transform [read], [nonatomic], [assign]
```

The color transform used to init the threshold filter

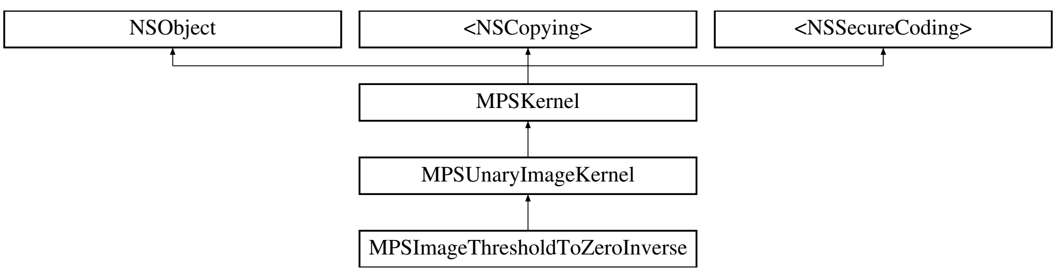
The documentation for this class was generated from the following file:

- MPSImageThreshold.h

5.113 MPSImageThresholdToZeroInverse Class Reference

```
#import <MPSImageThreshold.h>
```

Inheritance diagram for MPSImageThresholdToZeroInverse:



Instance Methods

- (nonnull instancetype) - initWithDevice:thresholdValue:linearGrayColorTransform:
- (nullable instancetype) - initWithCoder:device:
- (nonnull instancetype) - initWithDevice:

Properties

- float thresholdValue
- const float * transform

Additional Inherited Members

5.113.1 Detailed Description

The MPSImageThresholdToZeroInverse filter applies a fixed-level threshold to each pixel in the image. The threshold functions convert a single channel image to a binary image. If the input image is not a single channel image, convert the inputimage to a single channel luminance image using the linearGrayColorTransform and then apply the threshold. The ThresholdToZeroINverse function is: destinationPixelValue = sourcePixelValue > thresholdValue ? 0 : sourcePixelValue

5.113.2 Method Documentation

5.113.2.1 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSUnaryImageKernel.

5.113.2.2 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

a pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSUnaryImageKernel.

5.113.2.3 initWithDevice:thresholdValue:linearGrayColorTransform:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            thresholdValue:(float) thresholdValue
            linearGrayColorTransform:(const float *__nullable) transform
```

initialize a MPSImageThresholdToZeroInverse filter

Parameters

device	The device the filter will run on
thresholdValue	The threshold value to use
transform	This matrix is an array of 3 floats. The default if no transform is specifed is BT.601/JPEG: {0.299f, 0.587f, 0.114f};

5.113.3 Property Documentation

5.113.3.1 thresholdValue

```
- thresholdValue  [read], [nonatomic], [assign]
```

The threshold value used to init the threshold filter

5.113.3.2 transform

```
- transform [read], [nonatomic], [assign]
```

The color transform used to init the threshold filter

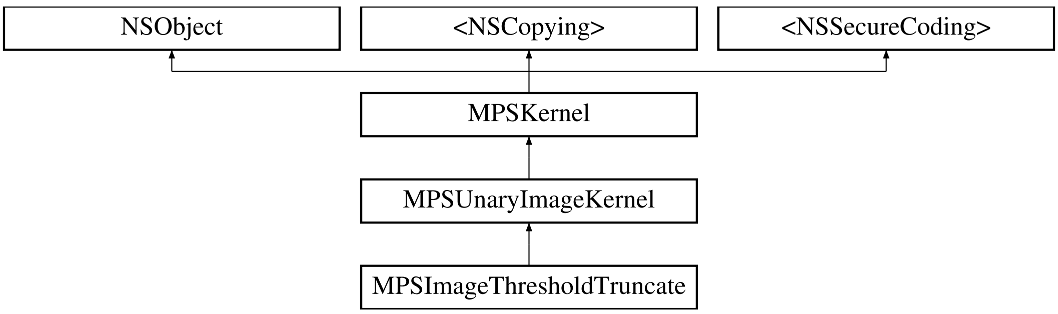
The documentation for this class was generated from the following file:

- MPSImageThreshold.h

5.114 MPSImageThresholdTruncate Class Reference

```
#import <MPSImageThreshold.h>
```

Inheritance diagram for MPSImageThresholdTruncate:



Instance Methods

- (nonnull instancetype) - initWithDevice:thresholdValue:linearGrayColorTransform:
- (nullable instancetype) - initWithCoder:device:
- (nonnull instancetype) - initWithDevice:

Properties

- float thresholdValue
- const float ∗ transform

Additional Inherited Members

5.114.1 Detailed Description

The MPSImageThresholdTruncate filter applies a fixed-level threshold to each pixel in the image: The threshold functions convert a single channel image to a binary image. If the input image is not a single channel image, convert the inputimage to a single channel luminance image using the linearGrayColorTransform and then apply the threshold. The ThresholdTruncate function is: destinationPixelValue = sourcePixelValue $>$ thresholdValue ? thresholdValue : sourcePixelValue

5.114.2 Method Documentation

5.114.2.1 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSUnaryImageKernel.

5.114.2.2 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

a pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSUnaryImageKernel.

5.114.2.3 initWithDevice:thresholdValue:linearGrayColorTransform:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            thresholdValue:(float) thresholdValue
            linearGrayColorTransform:(const float *__nullable) transform
```

initialize a MPSImageThresholdTruncate filter

Parameters

device	The device the filter will run on
thresholdValue	The threshold value to use
transform	This matrix is an array of 3 floats. The default if no transform is specifed is BT.601/JPEG: {0.299f, 0.587f, 0.114f};

5.114.3 Property Documentation

5.114.3.1 thresholdValue

```
- thresholdValue [read], [nonatomic], [assign]
```

The threshold value used to init the threshold filter

5.114.3.2 transform

```
- transform [read], [nonatomic], [assign]
```

The color transform used to init the threshold filter

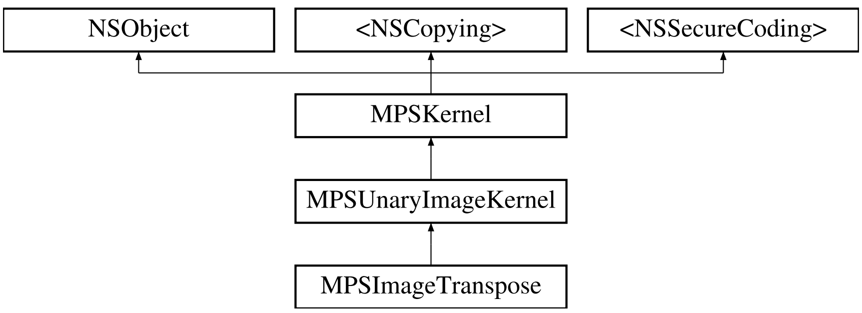
The documentation for this class was generated from the following file:

- MPSImageThreshold.h

5.115 MPSImageTranspose Class Reference

```
#import <MPSImageTranspose.h>
```

Inheritance diagram for MPSImageTranspose:



Additional Inherited Members

5.115.1 Detailed Description

MPSImageTranspose.h MetalPerformanceShaders.framework

Copyright

 MetalPerformanceShaders transpose filters

The MPSImageTranspose transposes an image

```
This kernel accepts uint and int textures in addition to unorm and floating-point textures.
```

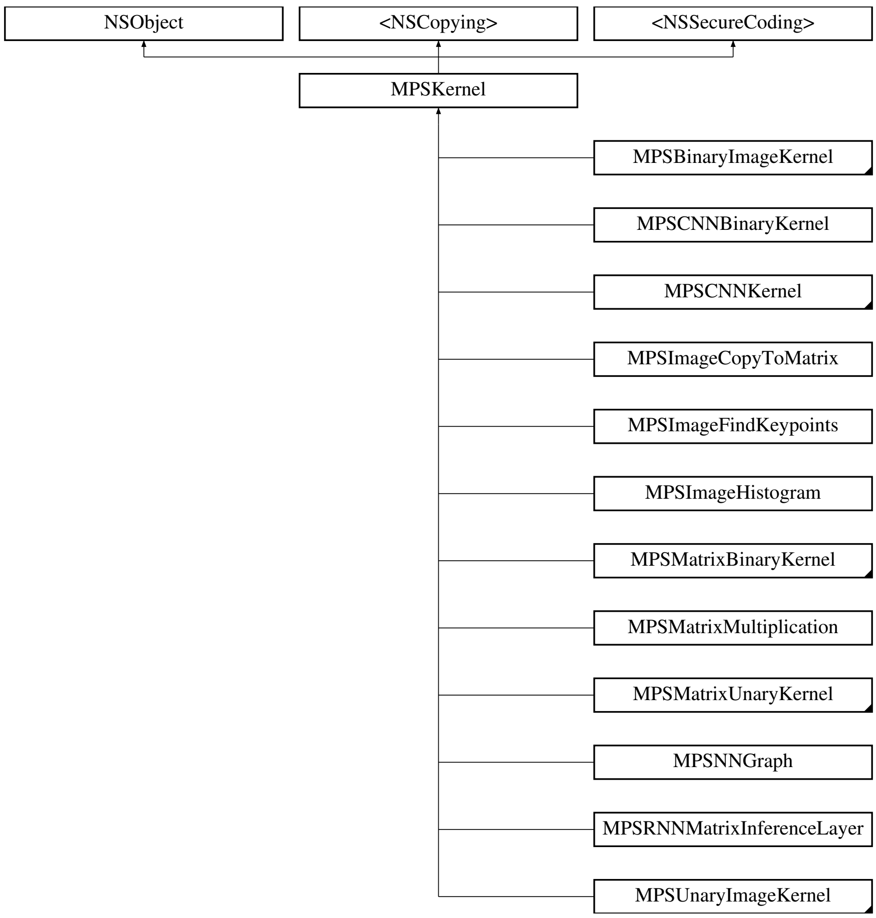
The documentation for this class was generated from the following file:

- MPSImageTranspose.h

5.116 MPSKernel Class Reference

```
#import <MPSKernel.h>
```

Inheritance diagram for MPSKernel:



Instance Methods

- (nonnull instancetype) - initWithDevice:
- (nonnull instancetype) - copyWithZone:device:
- (nullable instancetype) - initWithCoder:
- (nullable instancetype) - initWithCoder:device:
- () - MPSCopyAllocator

Public Attributes

- const MTLRegion MPSRectNoClip

Properties

- MPSKernelOptions options
- id< MTLDevice > device
- NSString * label

5.116.1 Detailed Description

MPSKernel.h MPSCore.framework

Copyright

MPSKernel objects encode tuned image processing operations into a MTLCommandBuffer.

This depends on Metal.framework The MPSKernel class is the base class for all MPS objects. It defines a standard interface for MPS kernels. You should not use the MPSKernel class directly. Instead, a number of MPSKernel subclasses are available in MetalPerformanceShaders.framework that define specific high-performance image processing operations.

The basic sequence for applying a MPSKernel to an image is as follows:

1. Create a MPSKernel corresponding to the operation you wish to perform:

```
MPSImageSobel *sobel = [[MPSImageSobel alloc] initWithDevice: mtlDevice];
```

2. Encode the filter into a command buffer:

```
sobel.offset = ...;
sobel.clipRect = ...;
sobel.options = ...;
[sobel encodeToCommandBuffer: commandBuffer
               sourceTexture: inputImage
          destinationTexture: resultImage ];
```

Encoding the kernel merely encodes the operation into a MTLCommandBuffer. It does not modify any pixels, yet. All MPSKernel state has been copied to the command buffer. MPSKernels may be reused. If the texture was previously operated on by another command encoder (e.g. MTLRenderCommandEncoder), you should call -endEncoding on the other encoder before encoding the filter.

Some MPS filters work in place (inputImage = resultImage) even in situations where Metal might not normally allow in place operation on textures. If in-place operation is desired, you may attempt to call [MPSKernel encodeKernelInPlace...]. If the operation can not be completed in place, then NO will be returned and you will have to create a new result texture and try again. To make an in-place image filter reliable, pass a fallback MPSCopyAllocator to the method to create a new texture to write to in the event that a filter can not operate in place.

(Repeat steps 2 for more filters, as desired.)

```
It should be self evident that step 2 may not be thread safe. That is, you can not have
multiple threads manipulating the same properties on the same MPSKernel object at the
same time and achieve coherent output. In common usage, the MPSKernel properties don't
often need to be changed from their default values, but if you need to apply the same
filter to multiple images on multiple threads with cropping / tiling, make additional
MPSKernel objects per thread. They are cheap. You can use multiple MPSKernel objects on
multiple threads, as long as only one thread is operating on any particular MPSKernel
object at a time.
```

3. After encoding any additional work to the command buffer using other encoders, submit the MTLCommand↩Buffer to your MTLCommandQueue, using:

```
[mtlCommandBuffer commit];
```

A MPSKernel can be saved to disk / network using NSCoders such as NSKeyedArchiver. When decoding, the system default MTLDevice will be chosen unless the NSCoder adopts the <MPSDeviceProvider> protocol. To accomplish this, subclass or extend your unarchiver to add this method.

5.116.2 Method Documentation

5.116.2.1 copyWithZone:device:()

```
- (nonnull instancetype) copyWithZone:
            (nullable NSZone *) zone
            device:(nullable id< MTLDevice >) device
```

Make a copy of this MPSKernel for a new device -copyWithZone: will call this API to make a copy of the MPSKernel on the same device. This interface may also be called directly to make a copy of the MPSKernel on a new device. Typically, the same MPSKernels should not be used to encode kernels on multiple command buffers from multiple threads. Many MPSKernels have mutable properties that might be changed by the other thread while this one is trying to encode. If you need to use a MPSKernel from multiple threads make a copy of it for each additional thread using -copyWithZone: or -copyWithZone:device:

Parameters

zone	The NSZone in which to allocate the object
device	The device for the new MPSKernel. If nil, then use self.device.

Returns

a pointer to a copy of this MPSKernel. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented in MPSRNNMatrixInferenceLayer, and MPSRNNImageInferenceLayer.

5.116.2.2 initWithCoder:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
```

Called by NSCoder to decode MPSKernels This isn't the right interface to decode a MPSKernel, but it is the one that NSCoder uses. To enable your NSCoder (e.g. NSKeyedUnarchiver) to set which device to use extend the object to adopt the MPSDeviceProvider protocol. Otherwise, the Metal system default device will be used.

5.116.2.3 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented in MPSCNNBinaryConvolution, MPSCNNBinaryFullyConnected, MPSCNNConvolutionTranspose, MPSRNNMatrixInferenceLayer, MPSCNNConvolution, MPSCNNFullyConnected, MPSRNNImageInference↩ Layer, MPSCNNNeuron, MPSImagePyramid, MPSCNNBinaryKernel, MPSBinaryImageKernel, MPSCNN↩ DilatedPoolingMax, MPSImageSobel, MPSCNNPoolingAverage, MPSCNNPoolingL2Norm, MPSCNNCross↩ ChannelNormalization, MPSCNNPooling, MPSCNNPoolingMax, MPSImageHistogramSpecification, MPSImage↩ ThresholdToZeroInverse, MPSCNNKernel, MPSImageThresholdToZero, MPSImageHistogramEqualization, M↩ PSCNNLocalContrastNormalization, MPSImageBox, MPSImageGaussianBlur, MPSUnaryImageKernel, MP↩ SImageStatisticsMean, MPSImageThresholdBinary, MPSImageThresholdTruncate, MPSImageDilate, MPS↩ ImageScale, MPSImageLanczosScale, MPSImageBilinearScale, MPSImageStatisticsMeanAndVariance, MP↩ SImageConvolution, MPSImageThresholdBinaryInverse, MPSImageHistogram, MPSCNNSpatialNormalization, MPSImageCopyToMatrix, MPSImageFindKeypoints, MPSImageStatisticsMinAndMax, MPSImageMedian, MPS↩ ImageAreaMax, and MPSNNGraph.

5.116.2.4 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

a pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented in MPSCNNBinaryConvolution, MPSCNNBinaryFullyConnected, MPSCNNKernel, MPSCNNConvolutionTranspose, MPSRNNMatrixInferenceLayer, MPSCNNConvolution, MPSCNNFullyConnected, MPSRNNImageInferenceLayer, MPSBinaryImageKernel, MPSImagePyramid, MPSImageSobel, MPSCNNNeuronELU, MPSCNNCrossChannelNormalization, MPSCNNPooling, MPSImageThresholdToZeroInverse, MPSCNNNeuronSoftPlus, MPSCNNNeuronSoftSign, MPSCNNBinaryKernel, MPSImageThresholdToZero, MPSCNNNeuronTanH, MPSCNNNeuronAbsolute, MPSCNNLocalContrastNormalization, MPSImageBox, MPSImageGaussianBlur, MPSUnaryImageKernel, MPSCNNNeuronHardSigmoid, MPSImageStatisticsMean, MPSImageThresholdBinary, MPSImageThresholdTruncate, MPSImageDilate, MPSImageScale, MPSCNNNeuronReLU, MPSCNNNeuronSigmoid, MPSImageLanczosScale, MPSImageBilinearScale, MPSImageStatisticsMeanAndVariance, MPSMatrixMultiplication, MPSMatrixVectorMultiplication, MPSImageArithmetic, MPSImageAdd, MPSImageSubtract, MPSImageMultiply, MPSImageDivide, MPSImageThresholdBinaryInverse, MPSCNNNeuronLinear, MPSCNNSpatialNormalization, MPSImageFindKeypoints, MPSCNNUpsampling, MPSImageStatisticsMinAndMax, MPSImageMedian, and MPSImageAreaMax.

5.116.2.5 MPSCopyAllocator()

```
- MPSCopyAllocator
```

MPSImageKernel.h MetalPerformanceShaders.framework

Copyright

A block to make a copy of sourceTexture for MPSKernels that can only execute out of place. Some MPSKernel objects may not be able to operate in place. When that occurs, and in-place operation is requested, MPS will call back to this block to get a new texture to return instead. To avoid spending long periods of time allocating pages to back the MTLTexture, the block should attempt to reuse textures. The texture returned from the MPSCopyAllocator will be returned instead of the sourceTexture from the MPSKernel method on return.

```
// A MPSCopyAllocator to handle cases where in-place operation fails.
MPSCopyAllocator myAllocator = ^id <MTLTexture>( MPSKernel * __nonnull filter,
                                                __nonnull id <MTLCommandBuffer> cmdBuf,
                                                __nonnull id <MTLTexture> sourceTexture)
{
    MTLPixelFormat format = sourceTexture.pixelFormat;  // FIXME: is this format writable?
    MTLTextureDescriptor *d = [MTLTextureDescriptor texture2DDescriptorWithPixelFormat: format
                                 width: sourceTexture.width
                                height: sourceTexture.height
                             mipmapped: NO];
    d.usage = MTLTextureUsageShaderRead | MTLTextureUsageShaderWrite;

    //FIXME: Allocating a new texture each time is slow. They take up to 1 ms each.
    //       There are not too many milliseconds in a video frame! You can recycle
    //       old textures (or MTLBuffers and make textures from them) and reuse
    //       the memory here.
    id <MTLTexture> result = [cmdBuf.device newTextureWithDescriptor: d];

    // FIXME: If there is any metadata associated with sourceTexture such as colorspace
    //        information, MTLResource.label, MTLResource.cpuCacheMode mode,
    //        MTLResource.MTLPurgeableState, etc., it may need to be similarly associated
    //        with the new texture to avoid losing your metadata.

    // FIXME: If filter.clipRect doesn't cover the entire image, you may need to copy
    //        pixels from sourceTexture to the new texture or regions of the new texture
    //        will be uninitialized. You can make a MTLCommandEncoder to encode work on
    //        the MTLCommandBuffer here to do that work, if necessary. It will be
    //        scheduled to run immediately before the MPSKernel work. Do not call
    //        [MTLCommandBuffer enqueue/commit/waitUntilCompleted/waitUntilScheduled]
    //        in the MPSCopyAllocator block. Make sure to call -endEncoding on the
    //        MTLCommandEncoder so that the MTLCommandBuffer has no active encoder
    //        before returning.

    // CAUTION: The next command placed on the MTLCommandBuffer after the MPSCopyAllocator
    //          returns is almost assuredly going to be encoded with a MTLComputeCommandEncoder.
    //          Creating any other type of encoder in the MPSCopyAllocator will probably cost
    //          an additional 0.5 ms of both CPU _AND_ GPU time (or more!) due to a double
    //          mode switch penalty.

    // CAUTION: If other objects (in addition to the caller of -encodeToCommandBuffer:inPlaceTexture:...)
    //          own a reference to sourceTexture, they may need to be notified that
    //          sourceTexture has been replaced so that they can release that resource
    //          and adopt the new texture.

    //          The reference to sourceTexture owned by the caller of
    //          -encodeToCommandBuffer:inPlaceTexture... will be released by
    //          -encodeToCommandBuffer:inPlaceTexture:... after the kernel is encoded if
    //          and only if the MPSCopyAllocator is called, and the operation is successfully
    //          encoded out of place.

    return result;
    // d is autoreleased
};
```

If nil is returned by the allocator, NO will be returned by the calling function.

When the MPSCopyAllocator is called, no MTLCommandEncoder is active on the commandBuffer. You may create a MTLCommandEncoder in the block to initialize the texture. Make sure to call -endEncoding on it before returning, if you do.

Parameters

filter	A valid pointer to the MPSKernel that is calling the MPSCopyAllocator. From it you can get the clipRect of the intended operation.
commandBuffer	A valid MTLCommandBuffer. It can be used to obtain the device against which to allocate the new texture. You may also enqueue operations on the commandBuffer to initialize the texture on a encoder allocated in the block. You may not submit, enqueue or wait for scheduling/completion of the command buffer.
sourceTexture	The texture that is providing the source image for the filter. You may wish to use its size and MTLPixelFormat for the new texture, but it is not requred.

Returns

A new valid MTLTexture to use as the destination for the MPSKernel. If the calling function succeeds, its texture parameter will be overwritten with a pointer to this texture. If the calling function fails (highly unlikely, except for user error) then the texture will be released before the calling function returns.

5.116.3 Member Data Documentation

5.116.3.1 MPSRectNoClip

```
- (const MTLRegion) MPSRectNoClip
```

MPSRectNoClip This is a special constant to indicate no clipping is to be done. The entire image will be used. This is the default clipping rectangle or the input extent for MPSKernels.

5.116.4 Property Documentation

5.116.4.1 device

```
- device  [read], [nonatomic], [retain]
```

The device on which the kernel will be used

5.116.4.2 label

```
- label  [read], [write], [atomic], [copy]
```

A string to help identify this object.

5.116.4.3 options

```
- options  [read], [write], [nonatomic], [assign]
```

The set of options used to run the kernel. MPSKernelOptions

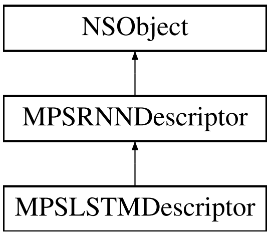
The documentation for this class was generated from the following files:

- MPSKernel.h
- MPSCoreTypes.h
- MPSImageKernel.h

5.117 MPSLSTMDescriptor Class Reference

```
#import <MPSRNNLayer.h>
```

Inheritance diagram for MPSLSTMDescriptor:



Class Methods

- (nonnull instancetype) + createLSTMDescriptorWithInputFeatureChannels:outputFeatureChannels:

Properties

- BOOL memoryWeightsAreDiagonal
- id< MPSCNNConvolutionDataSource > inputGateInputWeights
- id< MPSCNNConvolutionDataSource > inputGateRecurrentWeights
- id< MPSCNNConvolutionDataSource > inputGateMemoryWeights
- id< MPSCNNConvolutionDataSource > forgetGateInputWeights
- id< MPSCNNConvolutionDataSource > forgetGateRecurrentWeights
- id< MPSCNNConvolutionDataSource > forgetGateMemoryWeights
- id< MPSCNNConvolutionDataSource > outputGateInputWeights
- id< MPSCNNConvolutionDataSource > outputGateRecurrentWeights
- id< MPSCNNConvolutionDataSource > outputGateMemoryWeights
- id< MPSCNNConvolutionDataSource > cellGateInputWeights
- id< MPSCNNConvolutionDataSource > cellGateRecurrentWeights
- id< MPSCNNConvolutionDataSource > cellGateMemoryWeights
- MPSCNNNeuronType cellToOutputNeuronType
- float cellToOutputNeuronParamA
- float cellToOutputNeuronParamB

5.117.1 Detailed Description

This depends on Metal.framework The MPSLSTMDescriptor specifies a LSTM block/layer descriptor. The RNN layer initialized with a MPSLSTMDescriptor transforms the input data (image or matrix), the memory cell data and previous output with a set of filters, each producing one feature map in the output data and memory cell, according to the LSTM formulae detailed below. The user may provide the LSTM unit a single input or a sequence of inputs.

```
Description of operation:
```

Let x_j be the input data (at time index t of sequence, j index containing quadruplet: batch index, x,y and feature index (x=y=0 for matrices)). Let h0_j be the recurrent input (previous output) data from previous time step (at time index t-1 of sequence). Let h1_i be the output data produced at this time step. Let c0_j be the previous memory cell data (at time index t-1 of sequence). Let c1_i be the new memory cell data (at time index t-1 of sequence).

Let Wi_ij, Ui_ij, Vi_ij, be the input gate weights for input, recurrent input and memory cell (peephole) data respectively Let bi_i be the bias for the input gate

Let Wf_ij, Uf_ij, Vf_ij, be the forget gate weights for input, recurrent input and memory cell data respectively Let bf_i be the bias for the forget gate

Let Wo_ij, Uo_ij, Vo_ij, be the output gate weights for input, recurrent input and memory cell data respectively Let bo_i be the bias for the output gate

Let Wc_ij, Uc_ij, Vc_ij, be the memory cell gate weights for input, recurrent input and memory cell data respectively Let bc_i be the bias for the memory cell gate

Let gi(x), gf(x), go(x), gc(x) be neuron activation function for the input, forget, output gate and memory cell gate Let gh(x) be the activation function applied to result memory cell data

Then the new memory cell data c1_j and output image h1_i are computed as follows:

```
    I_i = gi(  Wi_ij * x_j  +  Ui_ij * h0_j  +  Vi_ij * c0_j  + bi_i  )
    F_i = gf(  Wf_ij * x_j  +  Uf_ij * h0_j  +  Vf_ij * c0_j  + bf_i  )
    C_i = gc(  Wc_ij * x_j  +  Uc_ij * h0_j  +  Vc_ij * c0_j  + bc_i  )

c1_i = F_i c0_i  +  I_i C_i

    O_i = go(  Wo_ij * x_j  +  Uo_ij * h0_j  +  Vo_ij * c1_j  + bo_i  )

h1_i = O_i gh( c1_i )
```

The '*' stands for convolution (see MPSRNNImageInferenceLayer) or matrix-vector/matrix multiplication (see MPSRNNMatrixInferenceLayer). Summation is over index j (except for the batch index), but there is no summation over repeated index i - the output index. Note that for validity all intermediate images have to be of same size and all U and V matrices have to be square (ie. outputFeatureChannels == inputFeatureChannels in those). Also the bias terms are scalars wrt. spatial dimensions.

5.117.2 Method Documentation

5.117.2.1 createLSTMDescriptorWithInputFeatureChannels:outputFeatureChannels:()

```
+ (nonnull instancetype) createLSTMDescriptorWithInputFeatureChannels:
            (NSUInteger) inputFeatureChannels
            outputFeatureChannels:(NSUInteger) outputFeatureChannels
```

Creates a LSTM descriptor.

Parameters

inputFeatureChannels	The number of feature channels in the input image/matrix. Must be >= 1.
outputFeatureChannels	The number of feature channels in the output image/matrix. Must be >= 1.

Returns

A valid MPSNNLSTMDescriptor object or nil, if failure.

5.117.3 Property Documentation

5.117.3.1 cellGateInputWeights

```
- cellGateInputWeights  [read], [write], [nonatomic], [retain]
```

Contains weights 'Wc_ij', bias 'bc_i' and neuron 'gc' from the LSTM formula. If nil then assumed zero weights, bias and no neuron (identity mapping). Defaults to nil.

5.117.3.2 cellGateMemoryWeights

```
- cellGateMemoryWeights  [read], [write], [nonatomic], [retain]
```

Contains weights 'Vc_ij' - the 'peephole' weights - from the LSTM formula. if YES == memoryWeightsAreDiagonal, then the number of weights used is the number of features in the memory cell image/matrix. If nil then assumed zero weights. Defaults to nil.

5.117.3.3 cellGateRecurrentWeights

```
- cellGateRecurrentWeights  [read], [write], [nonatomic], [retain]
```

Contains weights 'Uc_ij' from the LSTM formula. If nil then assumed zero weights. Defaults to nil.

5.117.3.4 cellToOutputNeuronParamA

```
- cellToOutputNeuronParamA  [read], [write], [nonatomic], [assign]
```

Neuron parameter A for 'gh'. Defaults to 1.0f.

5.117.3.5 cellToOutputNeuronParamB

```
- cellToOutputNeuronParamB  [read], [write], [nonatomic], [assign]
```

Neuron parameter B for 'gh'. Defaults to 1.0f.

5.117.3.6 cellToOutputNeuronType

```
- cellToOutputNeuronType  [read], [write], [nonatomic], [assign]
```

Neuron type definition for 'gh', see MPSCNNNeuronType. Defaults to MPSCNNNeuronTypeTanH.

5.117.3.7 forgetGateInputWeights

```
- forgetGateInputWeights  [read], [write], [nonatomic], [retain]
```

Contains weights 'Wf_ij', bias 'bf_i' and neuron 'gf' from the LSTM formula. If nil then assumed zero weights, bias and no neuron (identity mapping).Defaults to nil.

5.117.3.8 forgetGateMemoryWeights

```
- forgetGateMemoryWeights  [read], [write], [nonatomic], [retain]
```

Contains weights 'Vf_ij' - the 'peephole' weights - from the LSTM formula. if YES == memoryWeightsAreDiagonal, then the number of weights used is the number of features in the memory cell image/matrix. If nil then assumed zero weights. Defaults to nil.

5.117.3.9 forgetGateRecurrentWeights

```
- forgetGateRecurrentWeights  [read], [write], [nonatomic], [retain]
```

Contains weights 'Uf_ij' from the LSTM formula. If nil then assumed zero weights. Defaults to nil.

5.117.3.10 inputGateInputWeights

```
- inputGateInputWeights  [read], [write], [nonatomic], [retain]
```

Contains weights 'Wi_ij', bias 'bi_i' and neuron 'gi' from the LSTM formula. If nil then assumed zero weights, bias and no neuron (identity mapping). Defaults to nil.

5.117.3.11 inputGateMemoryWeights

```
- inputGateMemoryWeights  [read], [write], [nonatomic], [retain]
```

Contains weights 'Vi_ij' - the 'peephole' weights - from the LSTM formula. if YES == memoryWeightsAreDiagonal, then the number of weights used is the number of features in the memory cell image/matrix. If nil then assumed zero weights. Defaults to nil.

5.117.3.12 inputGateRecurrentWeights

```
- inputGateRecurrentWeights  [read], [write], [nonatomic], [retain]
```

Contains weights 'Ui_ij' from the LSTM formula. If nil then assumed zero weights. Defaults to nil.

5.117.3.13 memoryWeightsAreDiagonal

```
- memoryWeightsAreDiagonal  [read], [write], [nonatomic], [assign]
```

If YES, then the 'peephole' weight matrices will be diagonal matrices represented as vectors of length the number of features in memory cells, that will be multiplied pointwise with the peephole matrix or image in order to achieve the diagonal (nonmixing) update. Defaults to NO.

5.117.3.14 outputGateInputWeights

```
- outputGateInputWeights [read], [write], [nonatomic], [retain]
```

Contains weights 'Wo_ij', bias 'bo_i' and neuron 'go' from the LSTM formula. If nil then assumed zero weights, bias and no neuron (identity mapping). Defaults to nil.

5.117.3.15 outputGateMemoryWeights

```
- outputGateMemoryWeights [read], [write], [nonatomic], [retain]
```

Contains weights 'Vo_ij' - the 'peephole' weights - from the LSTM. if YES == memoryWeightsAreDiagonal, then the number of weights used is the number of features in the memory cell image/matrix. If nil then assumed zero weights. Defaults to nil.

5.117.3.16 outputGateRecurrentWeights

```
- outputGateRecurrentWeights [read], [write], [nonatomic], [retain]
```

Contains weights 'Uo_ij' from the LSTM formula. If nil then assumed zero weights. Defaults to nil.

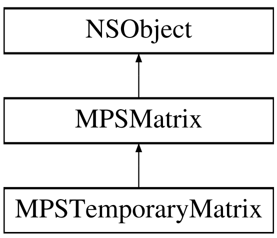
The documentation for this class was generated from the following file:

- MPSRNNLayer.h

5.118 MPSMatrix Class Reference

```
#import <MPSMatrixTypes.h>
```

Inheritance diagram for MPSMatrix:



Instance Methods

- (nonnull instancetype) - initWithBuffer:descriptor:
- (nonnull instancetype) - init

Properties

- id< MTLDevice > device
- NSUInteger rows
- NSUInteger columns
- NSUInteger matrices
- MPSDataType dataType
- NSUInteger rowBytes
- NSUInteger matrixBytes
- id< MTLBuffer > data

5.118.1 Detailed Description

This depends on Metal.framework

A MPSMatrix object describes a set of 2-dimensional arrays of data and provides storage for its values. MPSMatrix objects serve as inputs and outputs of MPSMatrixKernel objects.

Implementation note: A MPSMatrix object maintains its internal storage using a MTLBuffer object and thus the same rules for maintaining coherency of a MTLBuffer's data between CPU memory and GPU memory apply to a MPSMatrix. An MPSMatrix object's data refers to an array of matrices. Data is assumed to be ordered by matrix first, followed by row, followed by column.

For example, index [i,j] of the k'th matrix of an MPSMatrix is located at byte offset: k $*$ matrixBytes + i $*$ rowBytes + j $*$ sizeof(dataType)

Where matrixBytes is a multiple of rowBytes at least equal to rows $*$ rowBytes.

5.118.2 Method Documentation

5.118.2.1 init()

```
- (nonnull instancetype) init
```

5.118.2.2 initWithBuffer:descriptor:()

```
- (nonnull instancetype) initWithBuffer:
            (nonnull id< MTLBuffer >) buffer
            descriptor:(nonnull MPSMatrixDescriptor *) descriptor
```

Initialize a MPSMatrix object with a MTLBuffer.

Parameters

buffer	The MTLBuffer object which contains the data to use for the MPSMatrix. May not be NULL.
descriptor	The MPSMatrixDescriptor. May not be NULL.

Returns

A valid MPSMatrix object or nil, if failure.

This function returns a MPSMatrix object which uses the supplied MTLBuffer. The dimensions and stride of the matrix are specified by the MPSMatrixDescriptor object.

The provided MTLBuffer must have enough storage to hold

```
(descriptor.matrices-1) * descriptor.matrixBytes +
(descriptor.rows-1) * descriptor.rowBytes +
 descriptor.columns * (element size) bytes.
```

Reimplemented in MPSTemporaryMatrix.

5.118.3 Property Documentation

5.118.3.1 columns

```
- columns  [read], [nonatomic], [assign]
```

The number of columns in a matrix in the MPSMatrix.

5.118.3.2 data

```
- data  [read], [nonatomic], [assign]
```

An MTLBuffer to store the data.

5.118.3.3 dataType

```
- dataType  [read], [nonatomic], [assign]
```

The type of the MPSMatrix data.

5.118.3.4 device

```
- device  [read], [nonatomic], [retain]
```

The device on which the MPSMatrix will be used.

5.118.3.5 matrices

```
- matrices  [read], [nonatomic], [assign]
```

The number of matrices in the MPSMatrix.

5.118.3.6 matrixBytes

```
- matrixBytes  [read], [nonatomic], [assign]
```

The stride, in bytes, between corresponding elements of consecutive matrices.

5.118.3.7 rowBytes

```
- rowBytes  [read], [nonatomic], [assign]
```

The stride, in bytes, between corresponding elements of consecutive rows.

5.118.3.8 rows

```
- rows  [read], [nonatomic], [assign]
```

The number of rows in a matrix in the MPSMatrix.

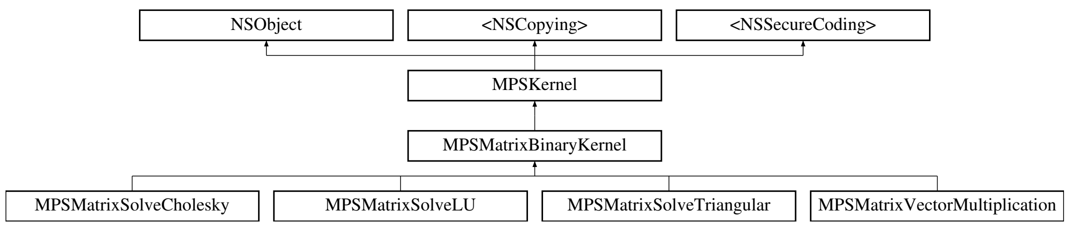
The documentation for this class was generated from the following file:

- MPSMatrixTypes.h

5.119 MPSMatrixBinaryKernel Class Reference

```
#import <MPSMatrixTypes.h>
```

Inheritance diagram for MPSMatrixBinaryKernel:



Properties

- MTLOrigin primarySourceMatrixOrigin
- MTLOrigin secondarySourceMatrixOrigin
- MTLOrigin resultMatrixOrigin
- NSUInteger batchStart
- NSUInteger batchSize

Additional Inherited Members

5.119.1 Detailed Description

This depends on Metal.framework A MPSMatrixBinaryKernel consumes two MPSMatrix objects and produces one MPSMatrix object.

5.119.2 Property Documentation

5.119.2.1 batchSize

```
- batchSize  [read], [write], [nonatomic], [assign]
```

The number of matrices in the batch to process. This property is modifiable and by default allows all matrices available at encoding time to be processed. If a single matrix should be processed set this value to 1.

5.119.2.2 batchStart

```
- batchStart  [read], [write], [nonatomic], [assign]
```

The index of the first matrix in the batch. This property is modifiable and defaults to 0 at initialization time. If batch processing should begin at a different matrix this value should be modified prior to encoding the kernel.

5.119.2.3 primarySourceMatrixOrigin

```
- primarySourceMatrixOrigin  [read], [write], [nonatomic], [assign]
```

The origin, relative to [0, 0] in the primary source matrix, at which to start reading values. This property is modifiable and defaults to [0, 0] at initialization time. If a different origin is desired then this should be modified prior to encoding the kernel. The z value must be 0.

5.119.2.4 resultMatrixOrigin

```
- resultMatrixOrigin  [read], [write], [nonatomic], [assign]
```

The origin, relative to [0, 0] in the result matrix, at which to start writing results. This property is modifiable and defaults to [0, 0] at initialization time. If a different origin is desired then this should be modified prior to encoding the kernel. The z value must be 0.

5.119.2.5 secondarySourceMatrixOrigin

```
- secondarySourceMatrixOrigin [read], [write], [nonatomic], [assign]
```

The origin, relative to [0, 0] in the secondary source matrix, at which to start reading values. This property is modifiable and defaults to [0, 0] at initialization time. If a different origin is desired then this should be modified prior to encoding the kernel. The z value must be 0.

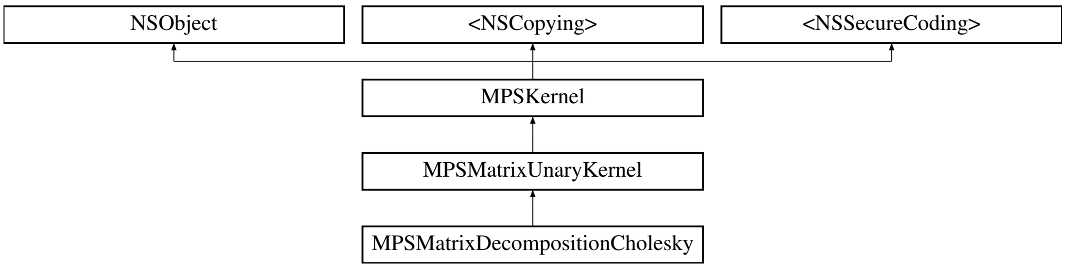
The documentation for this class was generated from the following file:

- MPSMatrixTypes.h

5.120 MPSMatrixDecompositionCholesky Class Reference

```
#import <MPSMatrixDecomposition.h>
```

Inheritance diagram for MPSMatrixDecompositionCholesky:



Instance Methods

- (nonnull instancetype) - initWithDevice:lower:order:
- (void) - encodeToCommandBuffer:sourceMatrix:resultMatrix:status:

Additional Inherited Members

5.120.1 Detailed Description

This depends on Metal.framework.

A kernel for computing the Cholesky factorization of a matrix.

A MPSMatrixDecompositionLU object computes one of the following factorizations of a matrix A:

```
A = L * L**T
A = U**T * U
```

A is a symmetric positive-definite matrix for which the factorization is to be computed. L and U are lower and upper triangular matrices respectively.

5.120.2 Method Documentation

5.120.2.1 encodeToCommandBuffer:sourceMatrix:resultMatrix:status:()

```
- (void) encodeToCommandBuffer:
            (nonnull id< MTLCommandBuffer >) commandBuffer
            sourceMatrix:(MPSMatrix const *__nonnull) sourceMatrix
            resultMatrix:(MPSMatrix *__nonnull) resultMatrix
            status:(__nullable id< MTLBuffer >) status
```

Encode a MPSMatrixDecompositionCholesky kernel into a command Buffer.

Parameters

commandBuffer	A valid MTLCommandBuffer to receive the encoded filter
sourceMatrix	A valid MPSMatrix containing the source data. Must have enough space to hold a order x order matrix.
resultMatrix	A valid MPSMatrix to contain the result. Must have enough space to hold a order x order matrix.
status	A MTLBuffer which indicates the resulting MPSMatrixDecompositionStatus value.

This function encodes the MPSMatrixDecompositionCholesky object to a valid command buffer.

If during the factorization a leading minor of the matrix is found to be not positive definite, MPSMatrix↩DecompositionNonPositiveDefinite will be returned in the provided status buffer. Previously computed pivots and the non positive pivot are written to the result, but the factorization does not complete. The data referenced by the MTLBuffer is not valid until the command buffer has completed execution. If the matrix return status is not desired NULL may be provided.

If the return status is MPSMatrixDecompositionStatusSuccess, resultMatrix contains the resulting factors in its lower or upper triangular regions respectively.

This kernel functions either in-place, if the result matrix completely aliases the source matrix, or out-of-place. If there is any partial overlap between input and output data the results are undefined.

5.120.2.2 initWithDevice:lower:order:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            lower:(BOOL) lower
            order:(NSUInteger) order
```

Initialize an MPSMatrixDecompositionCholesky object on a device

Parameters

device	The device on which the kernel will execute.
lower	A boolean value indicating if the lower triangular part of the source matrix is stored. If lower = YES the lower triangular part will be used and the factor will be written to the lower triangular part of the result, otherwise the upper triangular part will be used and the factor will be written to the upper triangular part.
order	The number of rows and columns in the source matrix.

Returns

A valid MPSMatrixDecompositionCholesky object or nil, if failure.

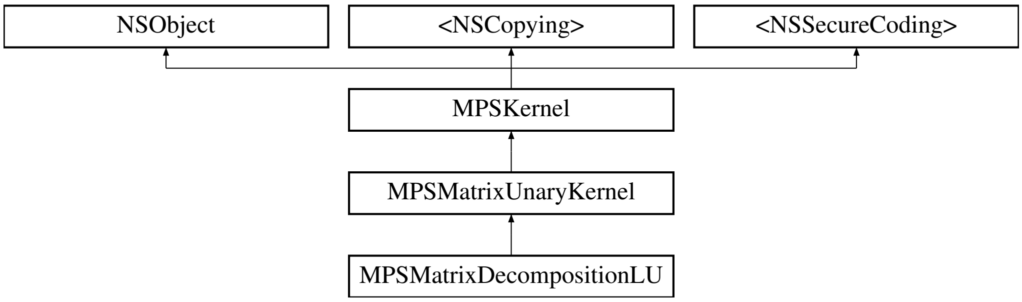
The documentation for this class was generated from the following file:

- MPSMatrixDecomposition.h

5.121 MPSMatrixDecompositionLU Class Reference

```
#import <MPSMatrixDecomposition.h>
```

Inheritance diagram for MPSMatrixDecompositionLU:



Instance Methods

- (nonnull instancetype) - initWithDevice:rows:columns:
- (void) - encodeToCommandBuffer:sourceMatrix:resultMatrix:pivotIndices:status:

Additional Inherited Members

5.121.1 Detailed Description

This depends on Metal.framework.

A kernel for computing the LU factorization of a matrix using partial pivoting with row interchanges.

A MPSMatrixDecompositionLU object computes an LU factorization:

```
    P * A = L * U

A is a matrix for which the LU factorization is to be computed.
L is a unit lower triangular matrix and U is an upper triangular
matrix.  P is a permutation matrix.
```

5.121.2 Method Documentation

5.121.2.1 encodeToCommandBuffer:sourceMatrix:resultMatrix:pivotIndices:status:()

```
- (void) encodeToCommandBuffer:
            (nonnull id< MTLCommandBuffer >) commandBuffer
            sourceMatrix:(MPSMatrix const *__nonnull) sourceMatrix
            resultMatrix:(MPSMatrix *__nonnull) resultMatrix
            pivotIndices:(MPSMatrix *__nonnull) pivotIndices
            status:(__nullable id< MTLBuffer >) status
```

Encode a MPSMatrixDecompositionLU kernel into a command Buffer.

Parameters

commandBuffer	A valid MTLCommandBuffer to receive the encoded filter
sourceMatrix	A valid MPSMatrix containing the source data. Must have enough space to hold a rows x columns matrix.
resultMatrix	A valid MPSMatrix to contain the result. Must have enough space to hold a rows x columns matrix.
pivotIndices	A valid MPSMatrix to contain the pivot indices. Must have enough space to hold an array of size 1xmin(rows, columns) values. Element type must be MPSDataTypeUInt32.
status	A MTLBuffer which indicates the resulting MPSMatrixDecompositionStatus value.

This function encodes the MPSMatrixDecompositionLU object to a valid command buffer.

Upon completion the array pivotIndices contains, for each index i, the row interchanged with row i.

If during the computation U[k, k], for some k, is determined to be exactly zero MPSMatrixDecompositionStatus←Singular will be returned in the provided status buffer. The data referenced by the MTLBuffer is not valid until the command buffer has completed execution. If the matrix return status is not desired NULL may be provided.

Upon successful factorization, resultMatrix contains the resulting lower triangular factor (without the unit diagonal elements) in its strictly lower triangular region and the upper triangular factor in its upper triangular region.

This kernel functions either in-place, if the result matrix completely aliases the source matrix, or out-of-place. If there is any partial overlap between input and output data the results are undefined.

5.121.2.2 initWithDevice:rows:columns:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            rows:(NSUInteger) rows
            columns:(NSUInteger) columns
```

Initialize an MPSMatrixDecompositionLU object on a device

Parameters

device	The device on which the kernel will execute.
rows	The number of rows in the source matrix.
columns	The number of columns in the source matrix.

Returns

A valid MPSMatrixDecompositionLU object or nil, if failure.

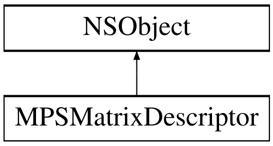
The documentation for this class was generated from the following file:

- MPSMatrixDecomposition.h

5.122 MPSMatrixDescriptor Class Reference

```
#import <MPSMatrixTypes.h>
```

Inheritance diagram for MPSMatrixDescriptor:



Class Methods

- (__nonnull instancetype) + matrixDescriptorWithDimensions:columns:rowBytes:dataType:
- (__nonnull instancetype) + matrixDescriptorWithRows:columns:rowBytes:dataType:
- (__nonnull instancetype) + matrixDescriptorWithRows:columns:matrices:rowBytes:matrixBytes:dataType:
- (size_t) + rowBytesFromColumns:dataType:
- (size_t) + rowBytesForColumns:dataType:

Properties

- NSUInteger rows
- NSUInteger columns
- NSUInteger matrices
- MPSDataType dataType
- NSUInteger rowBytes
- NSUInteger matrixBytes

5.122.1 Detailed Description

This depends on Metal.framework

A MPSMatrixDescriptor describes the sizes, strides, and data type of a an array of 2-dimensional matrices. All storage is assumed to be in "matrix-major". See the description for MPSMatrix for further details.

5.122.2 Method Documentation

5.122.2.1 matrixDescriptorWithDimensions:columns:rowBytes:dataType:()

```
+ (__nonnull instancetype) matrixDescriptorWithDimensions:
            (NSUInteger) rows
            columns:(NSUInteger) columns
            rowBytes:(NSUInteger) rowBytes
            dataType:(MPSDataType) dataType
```

Create a MPSMatrixDescriptor with the specified dimensions and data type.

Parameters

rows	The number of rows of the matrix.
columns	The number of columns of the matrix.
rowBytes	The number of bytes between starting elements of consecutive rows. Must be a multiple of the element size.
dataType	The type of the data to be stored in the matrix.

For performance considerations the optimal row stride may not necessarily be equal to the number of columns in the matrix. The MPSMatrix class provides a method which may be used to determine this value, see the row↩ BytesForColumns API in the MPSMatrix class. The number of matrices described is initialized to 1.

5.122.2.2 matrixDescriptorWithRows:columns:matrices:rowBytes:matrixBytes:dataType:()

```
+ (__nonnull instancetype) matrixDescriptorWithRows:
            (NSUInteger) rows
            columns:(NSUInteger) columns
            matrices:(NSUInteger) matrices
            rowBytes:(NSUInteger) rowBytes
            matrixBytes:(NSUInteger) matrixBytes
            dataType:(MPSDataType) dataType
```

Create a MPSMatrixDescriptor with the specified dimensions and data type.

Parameters

rows	The number of rows of a single matrix.
columns	The number of columns of a single matrix.
matrices	The number of matrices in the MPSMatrix object.
rowBytes	The number of bytes between starting elements of consecutive rows. Must be a multiple of the element size.
matrixBytes	The number of bytes between starting elements of consecutive matrices. Must be a multiple of rowBytes.
dataType	The type of the data to be stored in the matrix.

For performance considerations the optimal row stride may not necessarily be equal to the number of columns in the matrix. The MPSMatrix class provides a method which may be used to determine this value, see the row↩ BytesForColumns API in the MPSMatrix class.

5.122.2.3 matrixDescriptorWithRows:columns:rowBytes:dataType:()

```
+ (__nonnull instancetype) matrixDescriptorWithRows:
            (NSUInteger) rows
            columns:(NSUInteger) columns
            rowBytes:(NSUInteger) rowBytes
            dataType:(MPSDataType) dataType
```

5.122.2.4 rowBytesForColumns:dataType:()

```
+ (size_t) rowBytesForColumns:
             (NSUInteger) columns
             dataType:(MPSDataType) dataType
```

5.122.2.5 rowBytesFromColumns:dataType:()

```
+ (size_t) rowBytesFromColumns:
             (NSUInteger) columns
             dataType:(MPSDataType) dataType
```

Return the recommended row stride, in bytes, for a given number of columns.

Parameters

columns	The number of columns in the matrix for which the recommended row stride, in bytes, is to be determined.
dataType	The type of matrix data values.

To achieve best performance the optimal stride between rows of a matrix is not necessarily equivalent to the number of columns. This method returns the row stride, in bytes, which gives best performance for a given number of columns. Using this row stride to construct your array is recommended, but not required (provided that the stride used is still large enough to allocate a full row of data).

5.122.3 Property Documentation

5.122.3.1 columns

```
- columns [read], [write], [nonatomic], [assign]
```

The number of columns in a matrix.

5.122.3.2 dataType

```
- dataType [read], [write], [nonatomic], [assign]
```

The type of the data which makes up the values of the matrix.

5.122.3.3 matrices

```
- matrices [read], [nonatomic], [assign]
```

The number of matrices.

5.122.3.4 matrixBytes

```
- matrixBytes [read], [nonatomic], [assign]
```

The stride, in bytes, between corresponding elements of consecutive matrices. Must be a multiple of rowBytes.

5.122.3.5 rowBytes

```
- rowBytes [read], [write], [nonatomic], [assign]
```

The stride, in bytes, between corresponding elements of consecutive rows. Must be a multiple of the element size.

5.122.3.6 rows

```
- rows [read], [write], [nonatomic], [assign]
```

The number of rows in a matrix.

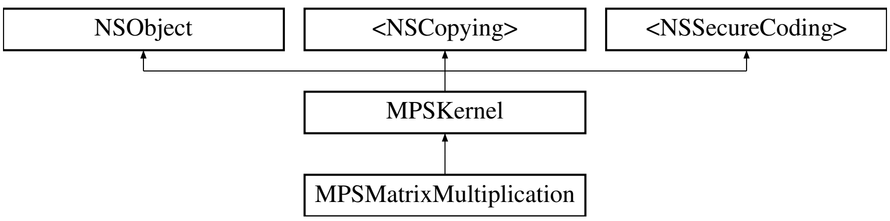
The documentation for this class was generated from the following file:

- MPSMatrixTypes.h

5.123 MPSMatrixMultiplication Class Reference

```
#import <MPSMatrixMultiplication.h>
```

Inheritance diagram for MPSMatrixMultiplication:



Instance Methods

- (nonnull instancetype) - initWithDevice:transposeLeft:transposeRight:resultRows:resultColumns:interiorColumns:alpha:beta:
- (nonnull instancetype) - initWithDevice:resultRows:resultColumns:interiorColumns:
- (nonnull instancetype) - initWithDevice:
- (void) - encodeToCommandBuffer:leftMatrix:rightMatrix:resultMatrix:

Properties

- MTLOrigin resultMatrixOrigin
- MTLOrigin leftMatrixOrigin
- MTLOrigin rightMatrixOrigin
- NSUInteger batchStart
- NSUInteger batchSize

Additional Inherited Members

5.123.1 Detailed Description

MPSMatrixMultiplication.h MetalPerformanceShaders.framework

Copyright

This depends on Metal.framework.

A matrix multiplication kernel.

A MPSMatrixMultiplication object computes:

```
    C = alpha * op(A) * op(B) + beta * C

A, B, and C are matrices which are represented by MPSMatrix
objects. alpha and beta are scalar values (of the same data type
as values of C) which are applied as shown above.  A and B may
each have an optional transposition operation applied.

A, B, and C (also referred to in later discussions as the left input
matrix, the right input matrix, and the result matrix respectively).

A MPSMatrixMultiplication object is initialized with the transpose
operators to apply to A and B, sizes for the operation to perform,
and the scalar values alpha and beta.
```

5.123.2 Method Documentation

5.123.2.1 encodeToCommandBuffer:leftMatrix:rightMatrix:resultMatrix:()

```
- (void) encodeToCommandBuffer:
             (nonnull id< MTLCommandBuffer >) commandBuffer
             leftMatrix:(MPSMatrix const *__nonnull) leftMatrix
             rightMatrix:(MPSMatrix const *__nonnull) rightMatrix
             resultMatrix:(MPSMatrix *__nonnull) resultMatrix
```

Encode a MPSMatrixMultiplication object to a command buffer.

Parameters

commandBuffer	A valid MTLCommandBuffer to receive the encoded kernel.
leftMatrix	A valid MPSMatrix object which specifies the left input matrix.
rightMatrix	A valid MPSMatrix object which specifies the right input matrix.
resultMatrix	A valid MPSMatrix object which specifies the addend matrix which will also be overwritten by the result.

Certain constraints apply to the sizes of the matrices depending on the transposition operations and sizes requested at initialization time as well as the origins at the time this routine is called:

The left input matrix must be large enough to hold an array of size resultRows x interiorColumns elements beginning at leftMatrixOrigin.

The right input matrix must be large enough to hold an array of size interiorColumns x resultColumns elements beginning at rightMatrixOrigin.

The result matrix must be large enough to hold an array of size resultRows x resultColumns elements beginning at resultMatrixOrigin.

Each matrix within the range specified by batchStart and batchSize, which also specifies a valid set of matrices within leftMatrix, rightMatrix, and resultMatrix, will be processed.

5.123.2.2 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Use the above initialization method instead.

Reimplemented from MPSKernel.

5.123.2.3 initWithDevice:resultRows:resultColumns:interiorColumns:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            resultRows:(NSUInteger) resultRows
            resultColumns:(NSUInteger) resultColumns
            interiorColumns:(NSUInteger) interiorColumns
```

Convenience initialization for a matrix-matrix multiplication with no transpositions, unit scaling of the product, and no accumulation of the result. The scaling factors alpha and beta are taken to be 1.0 and 0.0 respectively.

Parameters

device	The device on which the kernel will execute.
resultRows	The number of rows in the result matrix, M in BLAS GEMM description.
resultColumns	The number of columns in the result matrix, N in BLAS GEMM description.
interiorColumns	The number of columns of the left input matrix. K in BLAS GEMM description.

Returns

A valid MPSMatrixMultiplication object or nil, if failure.

5.123.2.4 initWithDevice:transposeLeft:transposeRight:resultRows:resultColumns:interiorColumns:alpha:beta:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            transposeLeft:(BOOL) transposeLeft
            transposeRight:(BOOL) transposeRight
            resultRows:(NSUInteger) resultRows
            resultColumns:(NSUInteger) resultColumns
            interiorColumns:(NSUInteger) interiorColumns
            alpha:(double) alpha
            beta:(double) beta
```

Initialize an MPSMatrixMultiplication object on a device for a given size and desired transpose and scale values.

Parameters

device	The device on which the kernel will execute.
transposeLeft	A boolean value which indicates if the left input matrix should be used in transposed form. If 'YES' then op(A) = A∗∗T, otherwise op(A) = A.
transposeRight	A boolean value which indicates if the right input matrix should be used in transposed form. If 'YES' then op(B) = B∗∗T, otherwise op(B) = B.
resultRows	The number of rows in the result matrix, M in BLAS GEMM description.
resultColumns	The number of columns in the result matrix, N in BLAS GEMM description.
interiorColumns	The number of columns of the left input matrix after the appropriate transpose operation has been applied. K in BLAS GEMM description.
alpha	The scale factor to apply to the product. Specified in double precision. Will be converted to the appropriate precision in the implementation subject to rounding and/or clamping as necessary.
beta	The scale factor to apply to the initial values of C. Specified in double precision. Will be converted to the appropriate precision in the implementation subject to rounding and/or clamping as necessary.

Returns

A valid MPSMatrixMultiplication object or nil, if failure.

5.123.3 Property Documentation

5.123.3.1 batchSize

```
- batchSize [read], [write], [nonatomic], [assign]
```

The number of matrices in the batch to process. This property is modifiable and by default allows all matrices available at encoding time to be processed.

5.123.3.2 batchStart

```
- batchStart [read], [write], [nonatomic], [assign]
```

The index of the first matrix in the batch. This property is modifiable and defaults to 0 at initialization time. If batch processing should begin at a different matrix this value should be modified prior to encoding the kernel.

5.123.3.3 leftMatrixOrigin

```
- leftMatrixOrigin [read], [write], [nonatomic], [assign]
```

The origin, relative to [0, 0] in the left input matrix, at which to start reading values. This property is modifiable and defaults to [0, 0] at initialization time. If a different origin is desired then this should be modified prior to encoding the kernel. The z value must be 0.

5.123.3.4 resultMatrixOrigin

```
- resultMatrixOrigin [read], [write], [nonatomic], [assign]
```

The origin, relative to [0, 0] in the result matrix, at which to start writing (and reading if necessary) results. This property is modifiable and defaults to [0, 0] at initialization time. If a different origin is desired then this should be modified prior to encoding the kernel. The z value must be 0.

5.123.3.5 rightMatrixOrigin

```
- rightMatrixOrigin [read], [write], [nonatomic], [assign]
```

The origin, relative to [0, 0] in the right input matrix, at which to start reading values. This property is modifiable and defaults to [0, 0] at initialization time. If a different origin is desired then this should be modified prior to encoding the kernel. The z value must be 0.

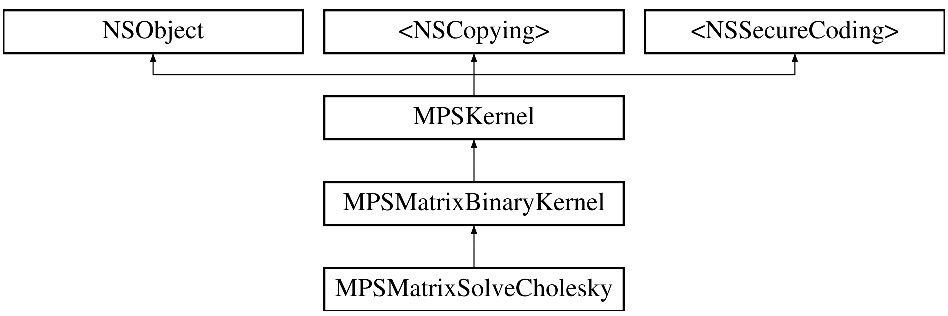
The documentation for this class was generated from the following file:

- MPSMatrixMultiplication.h

5.124 MPSMatrixSolveCholesky Class Reference

```
#import <MPSMatrixSolve.h>
```

Inheritance diagram for MPSMatrixSolveCholesky:



Instance Methods

- (nonnull instancetype) - initWithDevice:upper:order:numberOfRightHandSides:
- (void) - encodeToCommandBuffer:sourceMatrix:rightHandSideMatrix:solutionMatrix:

Additional Inherited Members

5.124.1 Detailed Description

This depends on Metal.framework.

A kernel for computing the solution of a linear system of equations using the Cholesky factorization resulting from a MPSMatrixDecompositionCholesky kernel.

A MPSMatrixSolveCholesky finds the solution matrix to the system:

```
    A * X = B

Where A is symmetric positive definite.  B is the array of
right hand sides for which the equations are to be solved.
X is the resulting matrix of solutions.
```

5.124.2 Method Documentation

5.124.2.1 encodeToCommandBuffer:sourceMatrix:rightHandSideMatrix:solutionMatrix:()

```
- (void) encodeToCommandBuffer:
            (nonnull id< MTLCommandBuffer >) commandBuffer
            sourceMatrix:(MPSMatrix const *__nonnull) sourceMatrix
            rightHandSideMatrix:(MPSMatrix const *__nonnull) rightHandSideMatrix
            solutionMatrix:(MPSMatrix *__nonnull) solutionMatrix
```

Encode a MPSMatrixSolveCholesky kernel into a command Buffer.

Parameters

commandBuffer	A valid MTLCommandBuffer to receive the encoded filter
sourceMatrix	A valid MPSMatrix containing the source matrix in factored form as returned by a previous successful execution of a MPSMatrixDecompositionCholesky kernel.
rightHandSideMatrix	A valid MPSMatrix containing the right hand side values.
solutionMatrix	A valid MPSMatrix to contain the result.

This function encodes the MPSMatrixSolveCholesky object to a valid command buffer. sourceMatrix should contain either the lower or upper triangular factors corresponding to the factorization returned by a previous execution of MPSMatrixDecompositionCholesky.

rightHandSideMatrix and solutionMatrix must be large enough to hold a matrix of size order x numberOfRight↵ HandSides starting at secondarySourceMatrixOrigin and resultMatrixOrigin respectively.

sourceMatrix must be at least size order x order starting at primarySourceMatrixOrigin.

5.124.2.2 initWithDevice:upper:order:numberOfRightHandSides:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            upper:(BOOL) upper
            order:(NSUInteger) order
            numberOfRightHandSides:(NSUInteger) numberOfRightHandSides
```

Initialize an MPSMatrixSolveCholesky object on a device

Parameters

device	The device on which the kernel will execute.
upper	A boolean value which indicates if the source matrix stores the lower or upper triangular factors.
order	The order of the source matrix and the number of rows in the solution and right hand side matrices.
numberOfRightHandSides	The number of columns in the solution and right hand side matrices.

Returns

A valid MPSMatrixSolveCholesky object or nil, if failure.

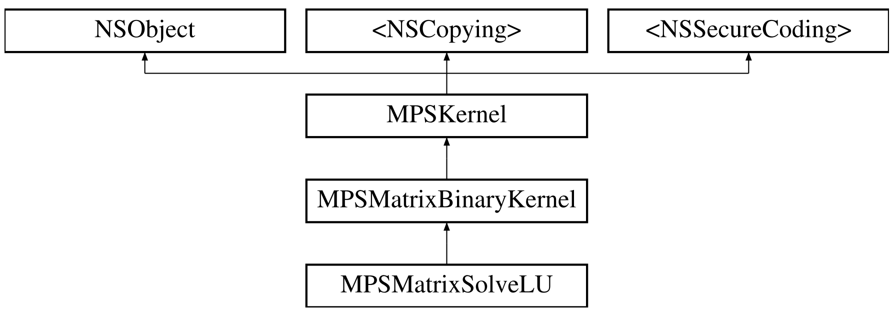
The documentation for this class was generated from the following file:

- MPSMatrixSolve.h

5.125 MPSMatrixSolveLU Class Reference

```
#import <MPSMatrixSolve.h>
```

Inheritance diagram for MPSMatrixSolveLU:



Instance Methods

- (nonnull instancetype) - initWithDevice:transpose:order:numberOfRightHandSides:
- (void) - encodeToCommandBuffer:sourceMatrix:rightHandSideMatrix:pivotIndices:solutionMatrix:

Additional Inherited Members

5.125.1 Detailed Description

This depends on Metal.framework.

A kernel for computing the solution of a linear system of equations using the LU factorization resulting from a MPSMatrixDecompositionLU kernel.

A MPSMatrixSolveLU finds the solution matrix to the system:

```
    op(A) * X = B

Where op(A) is A**T or A.  B is the array of right hand sides for which
the equations are to be solved.  X is the resulting matrix of solutions.
```

5.125.2 Method Documentation

5.125.2.1 encodeToCommandBuffer:sourceMatrix:rightHandSideMatrix:pivotIndices:solutionMatrix:()

```
- (void) encodeToCommandBuffer:
            (nonnull id< MTLCommandBuffer >) commandBuffer
            sourceMatrix:(MPSMatrix const *__nonnull) sourceMatrix
            rightHandSideMatrix:(MPSMatrix const *__nonnull) rightHandSideMatrix
            pivotIndices:(MPSMatrix const *__nonnull) pivotIndices
            solutionMatrix:(MPSMatrix *__nonnull) solutionMatrix
```

Encode a MPSMatrixSolveLU kernel into a command Buffer.

Parameters

commandBuffer	A valid MTLCommandBuffer to receive the encoded filter
sourceMatrix	A valid MPSMatrix containing the source matrix in factored form as returned by a previous successful execution of a MPSMatrixDecompositionLU kernel.
rightHandSideMatrix	A valid MPSMatrix containing the right hand side values.
pivotIndices	A valid MPSMatrix which contains the pivot indices as returned by a previous successful execution of a MPSMatrixDecompositionLU kernel.
solutionMatrix	A valid MPSMatrix to contain the result.

This function encodes the MPSMatrixSolveLU object to a valid command buffer. sourceMatrix should contain the lower and upper triangular factors of A as results from a previous execution of MPSMatrixDecompositionLU.

pivotIndices is an array of pivots resulting from a previous execution of MPSMatrixDecompositionLU.

rightHandSideMatrix and solutionMatrix must be large enough to hold a matrix of size order x numberOfRight↩HandSides starting at secondarySourceMatrixOrigin and resultMatrixOrigin respectively.

sourceMatrix must be at least size order x order starting at primarySourceMatrixOrigin.

5.125.2.2 initWithDevice:transpose:order:numberOfRightHandSides:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            transpose:(BOOL) transpose
            order:(NSUInteger) order
            numberOfRightHandSides:(NSUInteger) numberOfRightHandSides
```

Initialize an MPSMatrixSolveLU object on a device

Parameters

device	The device on which the kernel will execute.
transpose	A boolean value which indicates if the source matrix should be used in transposed form.
order	The order of the source matrix and the number of rows in the solution and right hand side matrices.
numberOfRightHandSides	The number of columns in the solution and right hand side matrices.

Returns

A valid MPSMatrixSolveLU object or nil, if failure.

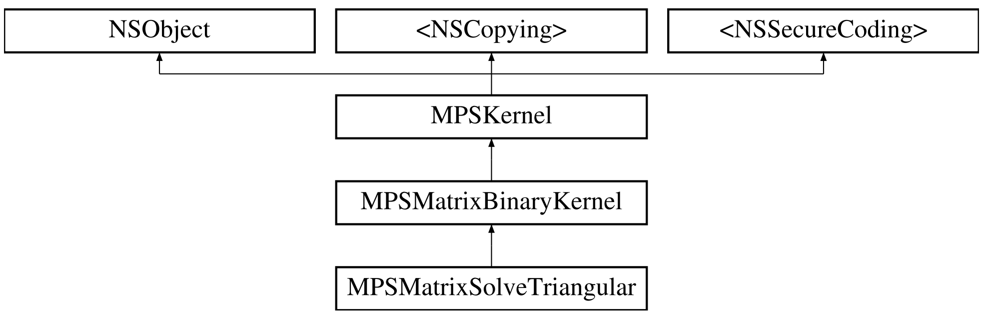
The documentation for this class was generated from the following file:

- MPSMatrixSolve.h

5.126 MPSMatrixSolveTriangular Class Reference

```
#import <MPSMatrixSolve.h>
```

Inheritance diagram for MPSMatrixSolveTriangular:



Instance Methods

- (nonnull instancetype) - initWithDevice:right:upper:transpose:unit:order:numberOfRightHandSides:alpha:
- (void) - encodeToCommandBuffer:sourceMatrix:rightHandSideMatrix:solutionMatrix:

Additional Inherited Members

5.126.1 Detailed Description

MPSMatrixSolve.h MetalPerformanceShaders.framework

Copyright

 MetalPerformanceShaders filter base classes

This depends on Metal.framework.

A kernel for computing the solution of a linear system of equations using a triangular coefficient matrix.

A MPSMatrixSolveTriangular finds the solution matrix to the triangular system:

```
op(A) * X = alpha * B    or    X * op(A) = alpha * B
```

Where A is either upper or lower triangular and op(A) is A**T or A. B is the array of right hand sides for which the equations are to be solved. X is the resulting matrix of solutions.

5.126.2 Method Documentation

5.126.2.1 encodeToCommandBuffer:sourceMatrix:rightHandSideMatrix:solutionMatrix:()

```
- (void) encodeToCommandBuffer:
            (nonnull id< MTLCommandBuffer >) commandBuffer
            sourceMatrix:(MPSMatrix const *__nonnull) sourceMatrix
            rightHandSideMatrix:(MPSMatrix const *__nonnull) rightHandSideMatrix
            solutionMatrix:(MPSMatrix *__nonnull) solutionMatrix
```

Encode a MPSMatrixSolveTriangular kernel into a command Buffer.

Parameters

commandBuffer	A valid MTLCommandBuffer to receive the encoded filter
sourceMatrix	A valid MPSMatrix containing the source matrix.
rightHandSideMatrix	A valid MPSMatrix containing the right hand side values.
solutionMatrix	A valid MPSMatrix to contain the result.

This function encodes the MPSMatrixSolveTriangular object to a valid command buffer.

rightHandSideMatrix and solutionMatrix must be large enough to hold at least order ∗ numberOfRightHandSides values starting at secondarySourceMatrixOrigin and resultMatrixOrigin respectively.

sourceMatrix must be at least size order x order starting at primarySourceMatrixOrigin.

5.126.2.2 initWithDevice:right:upper:transpose:unit:order:numberOfRightHandSides:alpha:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            right:(BOOL) right
            upper:(BOOL) upper
            transpose:(BOOL) transpose
            unit:(BOOL) unit
            order:(NSUInteger) order
            numberOfRightHandSides:(NSUInteger) numberOfRightHandSides
            alpha:(double) alpha
```

Initialize an MPSMatrixSolveTriangular object on a device

Parameters

device	The device on which the kernel will execute.
right	A boolean value which indicates if the coefficient matrix is multiplied on the left or right side of the solution. NO indicates the multiplication is on the left.
upper	A boolean value which indicates if the source is lower or upper triangular. NO indicates that the coefficient matrix is lower triangular.
transpose	A boolean value which indicates if the source matrix should be used in transposed form. NO indicates that the coefficient matrix is to be used normally.
unit	A boolean value which indicates if the source matrix is unit triangular.
order	The order of the source matrix and, if right == NO, the number of rows in the solution and right hand side matrices. If right == YES the number of columns in the solution and right hand side matrices.
numberOfRightHandSides	If right == NO, the number of columns in the solution and right hand side matrices. The number of rows otherwise.
alpha	A double precision value used to scale the right hand sides.

This function initializes a MPSMatrixSolveTriangular object. It may allocate device side memory.

Returns

A valid MPSMatrixSolveTriangular object or nil, if failure.

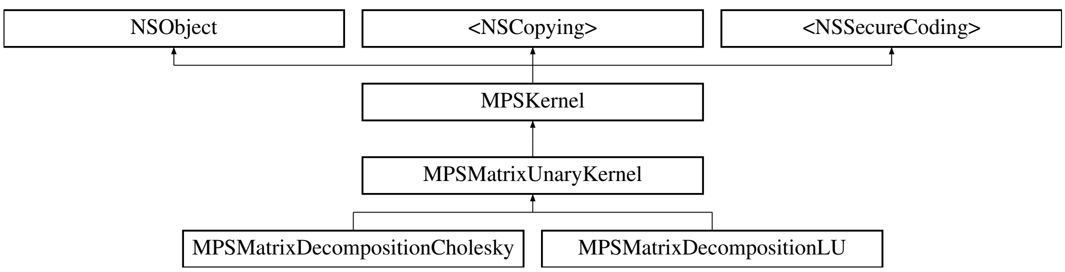
The documentation for this class was generated from the following file:

- MPSMatrixSolve.h

5.127 MPSMatrixUnaryKernel Class Reference

```
#import <MPSMatrixTypes.h>
```

Inheritance diagram for MPSMatrixUnaryKernel:



Properties

- MTLOrigin sourceMatrixOrigin
- MTLOrigin resultMatrixOrigin
- NSUInteger batchStart
- NSUInteger batchSize

Additional Inherited Members

5.127.1 Detailed Description

This depends on Metal.framework A MPSMatrixUnaryKernel consumes one MPSMatrix and produces one MPS←Matrix.

5.127.2 Property Documentation

5.127.2.1 batchSize

```
- batchSize  [read], [write], [nonatomic], [assign]
```

The number of matrices in the batch to process. This property is modifiable and by default allows all matrices available at encoding time to be processed. If a single matrix should be processed set this value to 1.

5.127.2.2 batchStart

```
- batchStart  [read], [write], [nonatomic], [assign]
```

The index of the first matrix in the batch. This property is modifiable and defaults to 0 at initialization time. If batch processing should begin at a different matrix this value should be modified prior to encoding the kernel.

5.127.2.3 resultMatrixOrigin

```
- resultMatrixOrigin  [read], [write], [nonatomic], [assign]
```

The origin, relative to [0, 0] in the result matrix, at which to start writing results. This property is modifiable and defaults to [0, 0] at initialization time. If a different origin is desired then this should be modified prior to encoding the kernel. The z value must be 0.

5.127.2.4 sourceMatrixOrigin

```
- sourceMatrixOrigin  [read], [write], [nonatomic], [assign]
```

The origin, relative to [0, 0] in the source matrix, at which to start reading values. This property is modifiable and defaults to [0, 0] at initialization time. If a different origin is desired then this should be modified prior to encoding the kernel. The z value must be 0.

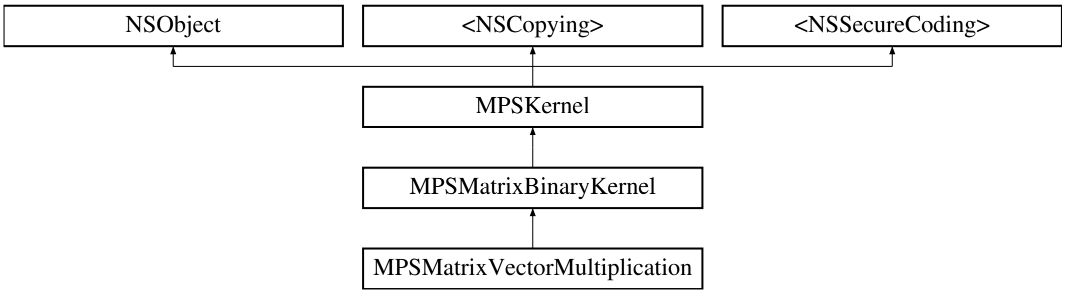
The documentation for this class was generated from the following file:

- MPSMatrixTypes.h

5.128 MPSMatrixVectorMultiplication Class Reference

```
#import <MPSMatrixMultiplication.h>
```

Inheritance diagram for MPSMatrixVectorMultiplication:



Instance Methods

- (nonnull instancetype) - initWithDevice:transpose:rows:columns:alpha:beta:
- (nonnull instancetype) - initWithDevice:rows:columns:
- (nonnull instancetype) - initWithDevice:
- (void) - encodeToCommandBuffer:inputMatrix:inputVector:resultVector:

Additional Inherited Members

5.128.1 Detailed Description

This depends on Metal.framework.

A matrix-vector multiplication kernel.

A MPSMatrixVectorMultiplication object computes:

```
        y = alpha * op(A) * x + beta * y

    A is a matrix represented by a MPSMatrix object. alpha and beta
    are scalar values (of the same data type as values of y) which are
    applied as shown above.  A may have an optional transposition
    operation applied.

    A MPSMatrixVectorMultiplication object is initialized with the transpose
    operator to apply to A, sizes for the operation to perform,
    and the scalar values alpha and beta.
```

5.128.2 Method Documentation

5.128.2.1 encodeToCommandBuffer:inputMatrix:inputVector:resultVector:()

```
- (void) encodeToCommandBuffer:
             (nonnull id< MTLCommandBuffer >) commandBuffer
             inputMatrix:(MPSMatrix const *__nonnull) inputMatrix
             inputVector:(MPSVector const *__nonnull) inputVector
             resultVector:(MPSVector *__nonnull) resultVector
```

Encode a MPSMatrixVectorMultiplication object to a command buffer.

Parameters

commandBuffer	A valid MTLCommandBuffer to receive the encoded kernel.
inputMatrix	A valid MPSMatrix object which specifies the input matrix A.
inputVector	A valid MPSVector object which specifies the input vector x.
resultVector	A valid MPSVector object which specifies the addend vector which will also be overwritten by the result.

The left input matrix must be large enough to hold an array of size (rows x columns) elements beginning at primary↩ SourceMatrixOrigin.

The input vector must be large enough to hold an array of size (columns) elements beginning at secondarySource↩ MatrixOrigin.x secondarySourceMatrixOrigin.y and secondarySourceMatrixOrigin.z must be zero.

The result vector must be large enough to hold an array of size (rows) elements beginning at resultMatrixOrigin.x. resultMatrixOrigin.y and resultMatrixOrigin.z must be zero.

5.128.2.2 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Use the above initialization method instead.

Reimplemented from MPSKernel.

5.128.2.3 initWithDevice:rows:columns:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            rows:(NSUInteger) rows
            columns:(NSUInteger) columns
```

Convenience initialization for a matrix-vector multiplication with no transposition, unit scaling of the product, and no accumulation of the result. The scaling factors alpha and beta are taken to be 1.0 and 0.0 respectively.

Parameters

device	The device on which the kernel will execute.
rows	The number of rows in the input matrix A, and the number of elements in the vector y.
columns	The number of columns in the input matrix A, and the number of elements in the input vector x.

Returns

A valid MPSMatrixVectorMultiplication object or nil, if failure.

5.128.2.4 initWithDevice:transpose:rows:columns:alpha:beta:()

```
- (nonnull instancetype) initWithDevice:
             (nonnull id< MTLDevice >) device
             transpose:(BOOL) transpose
             rows:(NSUInteger) rows
             columns:(NSUInteger) columns
             alpha:(double) alpha
             beta:(double) beta
```

Initialize an MPSMatrixVectorMultiplication object on a device for a given size and desired transpose and scale values.

Parameters

device	The device on which the kernel will execute.
transpose	A boolean value which indicates if the input matrix should be used in transposed form. if 'YES' then op(A) == A∗∗T, otherwise op(A) == A.
rows	The number of rows in the input matrix op(A), and the number of elements in the vector y.
columns	The number of columns in the input matrix op(A), and the number of elements in the input vector x.
alpha	The scale factor to apply to the product. Specified in double precision. Will be converted to the appropriate precision in the implementation subject to rounding and/or clamping as necessary.
beta	The scale factor to apply to the initial values of y. Specified in double precision. Will be converted to the appropriate precision in the implementation subject to rounding and/or clamping as necessary.

Returns

A valid MPSMatrixVectorMultiplication object or nil, if failure.

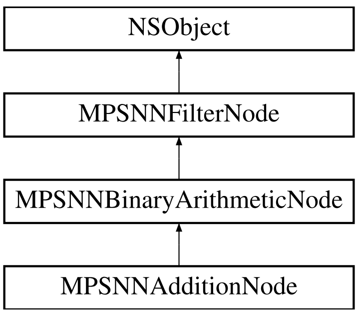
The documentation for this class was generated from the following file:

- MPSMatrixMultiplication.h

5.129 MPSNNAdditionNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSNNAdditionNode:



Additional Inherited Members

5.129.1 Detailed Description

returns elementwise sum of left + right

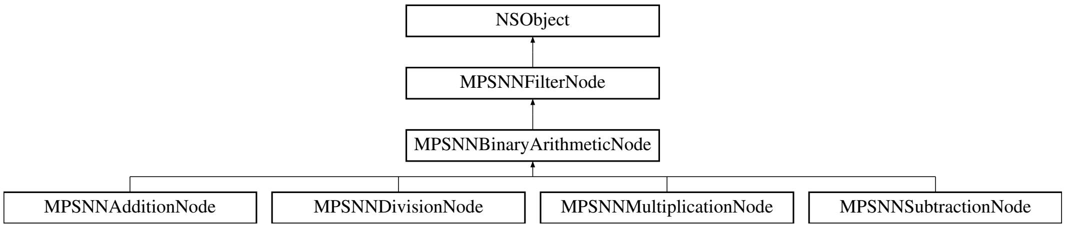
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.130 MPSNNBinaryArithmeticNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSNNBinaryArithmeticNode:



Instance Methods

- (nonnull instancetype) - initWithSources:
- (nonnull instancetype) - initWithLeftSource:rightSource:

Class Methods

- (nonnull instancetype) + nodeWithSources:
- (nonnull instancetype) + nodeWithLeftSource:rightSource:

Additional Inherited Members

5.130.1 Detailed Description

virtual base class for basic arithmetic nodes

5.130.2 Method Documentation

5.130.2.1 initWithLeftSource:rightSource:()

```
- (nonnull instancetype) initWithLeftSource:
            (MPSNNImageNode *__nonnull) left
            rightSource:(MPSNNImageNode *__nonnull) right
```

init an arithemtic node with two sources

Parameters

left	the left operand
right	the right operand

5.130.2.2 initWithSources:()

```
- (nonnull instancetype) initWithSources:
            (NSArray< MPSNNImageNode * > *__nonnull) sourceNodes
```

init an arithemtic node with an array of sources

Parameters

sourceNodes	A valid NSArray containing two sources

5.130.2.3 nodeWithLeftSource:rightSource:()

```
+ (nonnull instancetype) nodeWithLeftSource:
            (MPSNNImageNode *__nonnull) left
            rightSource:(MPSNNImageNode *__nonnull) right
```

create an autoreleased arithemtic node with two sources

Parameters

left	the left operand
right	the right operand

5.130.2.4 nodeWithSources:()

```
+ (nonnull instancetype) nodeWithSources:
            (NSArray< MPSNNImageNode * > *__nonnull) sourceNodes
```

create an autoreleased arithemtic node with an array of sources

Parameters

sourceNodes	A valid NSArray containing two sources

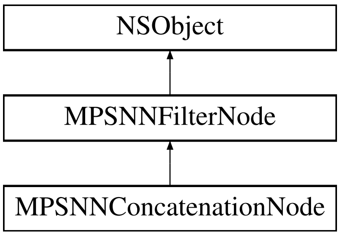
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.131 MPSNNConcatenationNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSNNConcatenationNode:



Instance Methods

- (nonnull instancetype) - initWithSources:

Class Methods

- (nonnull instancetype) + nodeWithSources:

Additional Inherited Members

5.131.1 Detailed Description

Node representing a the concatenation (in the feature channel dimension) of the results from one or more kernels

5.131.2 Method Documentation

5.131.2.1 initWithSources:()

```
- (nonnull instancetype) initWithSources:
            (NSArray< MPSNNImageNode * > *__nonnull) sourceNodes
```

Init a node that concatenates feature channels from multiple images In some neural network designs, it is necessary to append feature channels from one neural network filter to the results of another. If we have three image nodes with M, N and O feature channels in them, passed to -initWithSources: as @[imageM, imageN, imageO], then feature channels [0,M-1] will be drawn from image M, feature channels [M, M+N-1] will be drawn from image N and feature channels [M+N, M+N+O-1] will be drawn from image O.

As all images are padded out to a multiple of four feature channels, M, N and O here are also multiples of four, even when the MPSImages are not. That is, if the image is 23 feature channels and one channel of padding, it takes up 24 feature channels worth of space in the concatenated result.

Performance Note: Generally, concatenation is free as long as all of the sourceNodes are produced by filters in the same MPSNNGraph. Most MPSCNNKernels have the ability to write their results at a feature channel offset within a target MPSImage. However, if the MPSNNImageNode source nodes come from images external to the MPSNN↩ Graph, then we have to do a copy operation to assemble the concatenated node. As a result, when deciding where to break a large logical graph into multiple smaller MPSNNGraphs, it is better for concatenations to appear at the ends of subgraphs when possible rather than at the start, to the extent that all the images used in the concatenation are produced by that subgraph.

Parameters

sourceNodes	The MPSNNImageNode representing the source MPSImages for the filter

Returns

A new MPSNNFilter node that concatenates its inputs.

5.131.2.2 nodeWithSources:()

```
+ (nonnull instancetype) nodeWithSources:
            (NSArray< MPSNNImageNode * > *__nonnull) sourceNodes
```

Init a autoreleased node that concatenates feature channels from multiple images In some neural network designs, it is necessary to append feature channels from one neural network filter to the results of another. If we have three image nodes with M, N and O feature channels in them, passed to -initWithSources: as @[imageM, imageN, imageO], then feature channels [0,M-1] will be drawn from image M, feature channels [M, M+N-1] will be drawn from image N and feature channels [M+N, M+N+O-1] will be drawn from image O.

As all images are padded out to a multiple of four feature channels, M, N and O here are also multiples of four, even when the MPSImages are not. That is, if the image is 23 feature channels and one channel of padding, it takes up 24 feature channels worth of space in the concatenated result.

Performance Note: Generally, concatenation is free as long as all of the sourceNodes are produced by filters in the same MPSNNGraph. Most MPSCNNKernels have the ability to write their results at a feature channel offset within a target MPSImage. However, if the MPSNNImageNode source nodes come from images external to the MPSNN←Graph, then we have to do a copy operation to assemble the concatenated node. As a result, when deciding where to break a large logical graph into multiple smaller MPSNNGraphs, it is better for concatenations to appear at the ends of subgraphs when possible rather than at the start, to the extent that all the images used in the concatenation are produced by that subgraph.

Parameters

sourceNodes	The MPSNNImageNode representing the source MPSImages for the filter

Returns

A new MPSNNFilter node that concatenates its inputs.

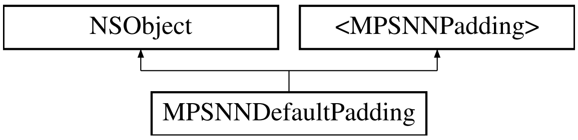
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.132 MPSNNDefaultPadding Class Reference

```
#import <MPSNeuralNetworkTypes.h>
```

Inheritance diagram for MPSNNDefaultPadding:



Instance Methods

- (NSString *__nonnull) - label

Class Methods

- (instancetype __nonnull) + paddingWithMethod:
- (instancetype __nonnull) + paddingForTensorflowAveragePooling

5.132.1 Method Documentation

5.132.1.1 label()

```
- (NSString * __nonnull) label
```

Human readable description of what the padding policy does

5.132.1.2 paddingForTensorflowAveragePooling()

```
+ (instancetype __nonnull) paddingForTensorflowAveragePooling
```

A padding policy that attempts to reproduce TensorFlow behavior for average pooling Most TensorFlow padding is covered by the standard MPSNNPaddingMethod encodings. You can use +paddingWithMethod to get quick access to MPSNNPadding objects, when default filter behavior isn't enough. (It often is.) However, the edging for max pooling in TensorFlow is a bit unusual.

This padding method attempts to reproduce TensorFlow padding for average pooling. In addition to setting MPSNNPaddingMethodSizeSame | MPSNNPaddingMethodAlignCentered | MPSNNPaddingMethodAddRemainderToTopLeft, it also configures the filter to run with MPSImageEdgeModeClamp, which (as a special case for average pooling only), normalizes the sum of contributing samples to the area of valid contributing pixels only.

```
// Sample implementation for the tensorflowPoolingPaddingPolicy returned
 -(MPSNNPaddingMethod) paddingMethod{ return
      MPSNNPaddingMethodCustom | MPSNNPaddingMethodSizeSame; }

 -(MPSImageDescriptor * __nonnull) destinationImageDescriptorForSourceImages: (NSArray
       <MPSImage *> *__nonnull) sourceImages
                                                                    sourceStates: (NSArray <MPSState *> *
      __nullable) sourceStates
                                                                       forKernel: (
      MPSKernel * __nonnull) kernel
                                                              suggestedDescriptor: (
      MPSImageDescriptor * __nonnull) inDescriptor
 {

    ((MPSCNNKernel *)kernel).edgeMode = MPSImageEdgeModeClamp;

    return inDescriptor;
 }
```

5.132.1.3 paddingWithMethod:()

```
+ (instancetype __nonnull) paddingWithMethod:
            (MPSNNPaddingMethod) method
```

Fetch a well known object that implements a non-custom padding method For custom padding methods, you will need to implement an object that conforms to the full MPSNNPadding protocol, including NSSecureCoding.

Parameters

method	A MPSNNPaddingMethod

Returns

An object that implements <MPSNNPadding> for use with MPSNNGraphNodes.

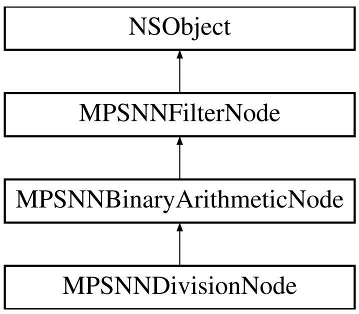
The documentation for this class was generated from the following file:

- MPSNeuralNetworkTypes.h

5.133 MPSNNDivisionNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSNNDivisionNode:



Additional Inherited Members

5.133.1 Detailed Description

returns elementwise quotient of left / right

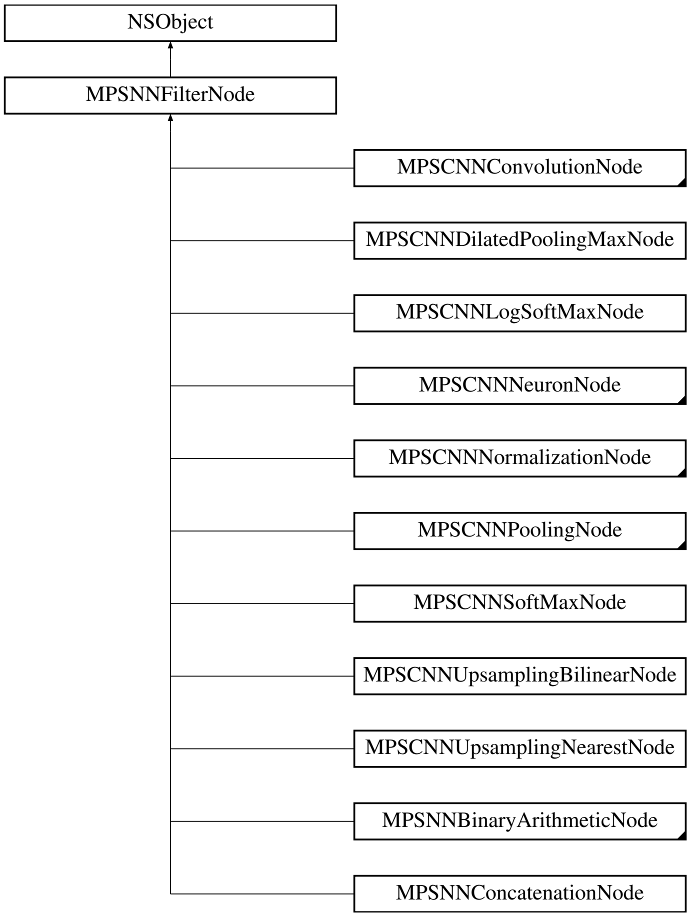
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.134 MPSNNFilterNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSNNFilterNode:



Instance Methods

- (nonnull instancetype) - init

Properties

- MPSNNImageNode ∗ resultImage
- MPSNNStateNode ∗ resultState
- NSArray< MPSNNStateNode ∗ > ∗ resultStates
- id< MPSNNPadding > paddingPolicy
- NSString ∗ label

5.134.1 Detailed Description

A placeholder node denoting a neural network filter stage There are as many MPSNNFilterNode subclasses as there are MPS neural network filter objects. Make one of those. This class defines an polymorphic interface for them.

5.134.2 Method Documentation

5.134.2.1 init()

```
- (nonnull instancetype) init
```

5.134.3 Property Documentation

5.134.3.1 label

```
- label [read], [write], [atomic], [copy]
```

A string to help identify this object.

5.134.3.2 paddingPolicy

```
- (id<MPSNNPadding>) paddingPolicy [read], [write], [nonatomic], [retain]
```

The padding method used for the filter node The default value varies per filter.

5.134.3.3 resultImage

```
- (MPSNNImageNode*) resultImage [read], [nonatomic], [assign]
```

Get the node representing the image result of the filter Except where otherwise noted, the precision used for the result image (see format property) is copied from the precision from the first input image node.

5.134.3.4 resultState

```
- (MPSNNStateNode*) resultState [read], [nonatomic], [assign]
```

convenience method for resultStates[0] If resultStates is nil, returns nil

5.134.3.5 resultStates

```
- (NSArray<MPSNNStateNode*>*) resultStates [read], [nonatomic], [assign]
```

Get the node representing the state result of the filter If more than one, see description of subclass for ordering.

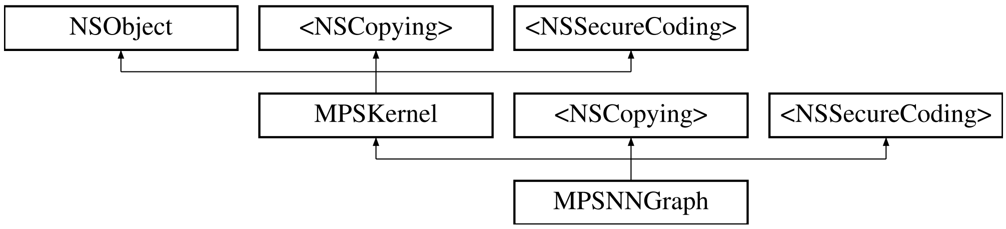
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.135 MPSNNGraph Class Reference

```
#import <MPSNNGraph.h>
```

Inheritance diagram for MPSNNGraph:



Instance Methods

- (nullable instancetype) - initWithDevice:resultImage:
- (nullable instancetype) - initWithCoder:device:
- (nonnull instancetype) - initWithDevice:
- (MPSImage ∗__nonnull) - encodeToCommandBuffer:sourceImages:sourceStates:intermediateImages↩ :destinationStates:
- (MPSImage ∗__nonnull) - encodeToCommandBuffer:sourceImages:
- (MPSImage ∗__nonnull) - executeAsyncWithSourceImages:completionHandler:

Properties

- NSArray< id< MPSHandle > > ∗ sourceImageHandles
- NSArray< id< MPSHandle > > ∗ sourceStateHandles
- NSArray< id< MPSHandle > > ∗ intermediateImageHandles
- NSArray< id< MPSHandle > > ∗ resultStateHandles
- id< MPSHandle > resultHandle
- BOOL outputStateIsTemporary
- id< MPSImageAllocator > destinationImageAllocator

Additional Inherited Members

5.135.1 Detailed Description

Optimized representation of a graph of MPSNNImageNodes and MPSNNFilterNodes Once you have prepared a graph of MPSNNImageNodes and MPSNNFilterNodes (and if needed MPSNNStateNodes), you may initialize a MPSNNGraph using the MPSNNImageNode that you wish to appear as the result. The MPSNNGraph object will introspect the graph representation and determine which nodes are needed for inputs, and which nodes are produced as output state (if any). Nodes which are not needed to calculate the result image node are ignored. Some nodes may be internally concatenated with other nodes for better performance.

Note: the MPSNNImageNode that you choose as the result node may be interior to a graph. This feature is provided as a means to examine intermediate computations in the full graph for debugging purposes.

During MPSNNGraph construction, the graph attached to the result node will be parsed and reduced to an optimized representation. This representation may be saved using the NSSecureCoding protocol for later recall.

When decoding a MPSNNGraph using a NSCoder, it will be created against the system default MTLDevice. If you would like to set the MTLDevice, your NSCoder should conform to the <MPSDeviceProvider> protocol.

You may find it helpful to set MPSKernelOptionsVerbose on the graph when debugging.

5.135.2 Method Documentation

5.135.2.1 encodeToCommandBuffer:sourceImages:()

```
- (MPSImage * __nonnull) encodeToCommandBuffer:
            (nonnull id< MTLCommandBuffer >) commandBuffer
            sourceImages:(NSArray< MPSImage * > *__nonnull) sourceImages
```

Encode the graph to a MTLCommandBuffer

IMPORTANT: Please use [MTLCommandBuffer addCompletedHandler:] to determine when this work is done. Use CPU time that would have been spent waiting for the GPU to encode the next command buffer and commit it too. That way, the work for the next command buffer is ready to go the moment the GPU is done. This will keep the GPU busy and running at top speed.

Those who ignore this advice and use [MTLCommandBuffer waitUntilCompleted] instead will likely cause their code to slow down by a factor of two or more. The CPU clock spins down while it waits for the GPU. When the GPU completes, the CPU runs slowly for a while until it spins up. The GPU has to wait for the CPU to encode more work (at low clock), giving it plenty of time to spin its own clock down. In typical CNN graph usage, neither may ever reach maximum clock frequency, causing slow down far beyond what otherwise would be expected from simple failure to schedule CPU and GPU work concurrently. Regrattably, it is probable that every performance benchmark you see on the net will be based on [MTLCommandBuffer waitUntilCompleted].

Parameters

commandBuffer	The command buffer
sourceImages	A list of MPSImages to use as the source images for the graph. These should be in the same order as the list returned from MPSNNGraph.sourceImageHandles.

Returns

A MPSImage or MPSTemporaryImage allocated per the destinationImageAllocator containing the output of the graph. It will be automatically released when commandBuffer completes.

5.135.2.2 encodeToCommandBuffer:sourceImages:sourceStates:intermediateImages:destinationStates:()

```
- (MPSImage * __nonnull) encodeToCommandBuffer:
            (nonnull id< MTLCommandBuffer >) commandBuffer
            sourceImages:(NSArray< MPSImage * > *__nonnull) sourceImages
            sourceStates:(NSArray< MPSState * > *__nullable) sourceStates
            intermediateImages:(NSMutableArray< MPSImage * > *__nullable) intermediateImages
            destinationStates:(NSMutableArray< MPSState * > *__nullable) destinationStates
```

Encode the graph to a MTLCommandBuffer IMPORTANT: Please use [MTLCommandBuffer addCompleted↩Handler:] to determine when this work is done. Use CPU time that would have been spent waiting for the GPU to encode the next command buffer and commit it too. That way, the work for the next command buffer is ready to go the moment the GPU is done. This will keep the GPU busy and running at top speed.

Those who ignore this advice and use [MTLCommandBuffer waitUntilCompleted] instead will likely cause their code to slow down by a factor of two or more. The CPU clock spins down while it waits for the GPU. When the GPU completes, the CPU runs slowly for a while until it spins up. The GPU has to wait for the CPU to encode more work (at low clock), giving it plenty of time to spin its own clock down. In typical CNN graph usage, neither may ever reach maximum clock frequency, causing slow down far beyond what otherwise would be expected from simple failure to schedule CPU and GPU work concurrently. Regrattably, it is probable that every performance benchmark you see on the net will be based on [MTLCommandBuffer waitUntilCompleted].

Parameters

commandBuffer	The command buffer
sourceImages	A list of MPSImages to use as the source images for the graph. These should be in the same order as the list returned from MPSNNGraph.sourceImageHandles. The images may be image arrays. Typically, this is only one or two images such as a .JPG decoded into a MPSImage∗. If the sourceImages are MPSTemporaryImages, the graph will decrement the readCount by 1, even if the graph actually reads an image multiple times.
sourceStates	A list of MPSState objects to use as state for a graph. These should be in the same order as the list returned from MPSNNGraph.sourceStateHandles. May be nil, if there is no source state. If the sourceStates are temporary, the graph will decrement the readCount by 1, even if the graph actually reads the state multiple times.
intermediateImages	An optional NSMutableArray to receive any MPSImage objects exported as part of its operation. These are only the images that were tagged with MPSNNImageNode.exportFromGraph = YES. The identity of the states is given by -resultStateHandles. If temporary, each intermediateImage will have a readCount of 1. If the result was tagged exportFromGraph = YES, it will be here too, with a readCount of 2.
destinationStates	An optional NSMutableArray to receive any MPSState objects created as part of its operation. The identity of the states is given by -resultStateHandles.

Returns

A MPSImage or MPSTemporaryImage allocated per the destinationImageAllocator containing the output of the graph. It will be automatically released when commandBuffer completes.

5.135.2.3 executeAsyncWithSourceImages:completionHandler:()

```
- (MPSImage * __nonnull) executeAsyncWithSourceImages:
            (NSArray< MPSImage * > *__nonnull) sourceImages
            completionHandler:(MPSNNGraphCompletionHandler __nonnull) handler
```

Convenience method to execute a graph without having to manage many Metal details This function will synchronously encode the graph on a private command buffer, commit it to a MPS internal command queue and return. The GPU will start working. When the GPU is done, the completion handler will be called. You should use the intervening time to encode other work for execution on the GPU, so that the GPU stays busy and doesn't clock down.

The work will be performed on the MTLDevice that hosts the source images.

This is a convenience API. There are a few situations it does not handle optimally. These may be better handled using [encodeToCommandBuffer:sourceImages:]. Specifically:

```
o     If the graph needs to be run multiple times for different images,
      it would be better to encode the graph multiple times on the same
      command buffer using [encodeToCommandBuffer:sourceImages:]  This
      will allow the multiple graphs to share memory for intermediate
      storage, dramatically reducing memory usage.

o     If preprocessing or post-processing of the MPSImage is required,
      such as resizing or normalization outside of a convolution, it would
      be better to encode those things on the same command buffer.
      Memory may be saved here too for intermediate storage. (MPSTemporaryImage
      lifetime does not span multiple command buffers.)
```

Parameters

sourceImages	A list of MPSImages to use as the source images for the graph. These should be in the same order as the list returned from MPSNNGraph.sourceImageHandles. They should be allocated against the same MTLDevice. There must be at least one source image. Note: this array is intended to handle the case where multiple input images are required to generate a single graph result. That is, the graph itself has multiple inputs. If you need to execute the graph multiple times, then call this API multiple times, or better yet use [encodeToCommandBuffer:sourceImages:] multiple times. (See discussion)
handler	A block to receive any errors generated. This block may run on any thread and may be called before this method returns. The image, if any, passed to this callback is the same image as that returned from the left hand side.

Returns

A MPSImage to receive the result. The data in the image will not be valid until the completionHandler is called.

5.135.2.4 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSKernel.

5.135.2.5 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (__nonnull id< MTLDevice >) device
```

Use initWithDevice:resultImage: instead

5.135.2.6 initWithDevice:resultImage:()

```
- (nullable instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            resultImage:(MPSNNImageNode *__nonnull) resultImage
```

Initialize a MPSNNGraph object on a device starting with resultImage working backward The MPSNNGraph constructor will start with the indicated result image, and look to see what MPSNNFilterNode produced it, then look to its dependencies and so forth to reveal the subsection of the graph necessary to compute the image.

Parameters

device	The MTLDevice on which to run the graph
resultImage	The MPSNNImageNode corresponding to the last image in the graph. This is the image that will be returned. Note: the imageAllocator for this node is ignored and the MPSNNGraph.destinationImageAllocator is used for this node instead.

Returns

A new MPSNNGraph.

5.135.3 Property Documentation

5.135.3.1 destinationImageAllocator

```
- (id<MPSImageAllocator>) destinationImageAllocator [read], [write], [nonatomic], [retain]
```

Method to allocate the result image from -encodeToCommandBuffer... This property overrides the allocator for the final result image in the graph. Default: defaultAllocator (MPSImage)

5.135.3.2 intermediateImageHandles

```
- (NSArray<id <MPSHandle> >*) intermediateImageHandles [read], [nonatomic], [copy]
```

Get a list of identifiers for intermediate images objects produced by the graph

5.135.3.3 outputStateIsTemporary

```
- (BOOL) outputStateIsTemporary [read], [write], [nonatomic], [assign]
```

Should MPSState objects produced by -encodeToCommandBuffer... be temporary objects. See MPSState description. Default: YES

5.135.3.4 resultHandle

```
- (id<MPSHandle>) resultHandle [read], [nonatomic], [assign]
```

Get a handle for the graph result image

5.135.3.5 resultStateHandles

```
- (NSArray<id <MPSHandle> >*) resultStateHandles [read], [nonatomic], [copy]
```

Get a list of identifiers for result state objects produced by the graph Not guaranteed to be in the same order as sourceStateHandles

5.135.3.6 sourceImageHandles

```
- (NSArray<id <MPSHandle> >*) sourceImageHandles [read], [nonatomic], [copy]
```

Get a list of identifiers for source images needed to calculate the result image

5.135.3.7 sourceStateHandles

```
- (NSArray<id <MPSHandle> >*) sourceStateHandles [read], [nonatomic], [copy]
```

Get a list of identifiers for source state objects needed to calculate the result image Not guaranteed to be in the same order as resultStateHandles

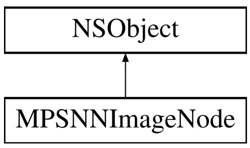
The documentation for this class was generated from the following file:

- MPSNNGraph.h

5.136 MPSNNImageNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSNNImageNode:



Instance Methods

- (nonnull instancetype) - initWithHandle:
- (nonnull instancetype) - init

Class Methods

- (nonnull instancetype) + nodeWithHandle:
- (nonnull instancetype) + exportedNodeWithHandle:

Properties

- id< MPSHandle > handle
- MPSImageFeatureChannelFormat format
- id< MPSImageAllocator > imageAllocator
- BOOL exportFromGraph

5.136.1 Detailed Description

A placeholder node denoting the position of a MPSImage in a graph MPS neural network graphs are made up of filter nodes connected by image (or state) nodes. An image node is produced by one filter but may be consumed by more than one filter.

Most image nodes will be created by MPS and made available through MPSNNFilterNode.resultImage. Image nodes that are not created by MPS (i.e. "the graph inputs") must be created by you.

5.136.2 Method Documentation

5.136.2.1 exportedNodeWithHandle:()

```
+ (nonnull instancetype) exportedNodeWithHandle:
            (NSObject< MPSHandle > *__nullable) handle
```

Create a autoreleased MPSNNImageNode with exportFromGraph = YES. Note: image is still temporary. See M↩ PSNNImageNode.imageAllocator parameter.

5.136.2.2 init()

```
- (nonnull instancetype) init
```

5.136.2.3 initWithHandle:()

```
- (nonnull instancetype) initWithHandle:
            (NSObject< MPSHandle > *__nullable) handle
```

5.136.2.4 nodeWithHandle:()

```
+ (nonnull instancetype) nodeWithHandle:
            (NSObject< MPSHandle > *__nullable) handle
```

5.136.3 Property Documentation

5.136.3.1 exportFromGraph

```
- (BOOL) exportFromGraph [read], [write], [nonatomic], [assign]
```

Tag a image node for view later Most image nodes are private to the graph. These alias memory heavily and consequently generally have invalid state when the graph exists. When exportFromGraph = YES, the image is preserved and made available through the [MPSNNGraph encode... intermediateImages:... list.

CAUTION: exporting an image from a graph prevents MPS from recycling memory. It will nearly always cause the amount of memory used by the graph to increase by the size of the image. There will probably be a performance regression accordingly. This feature should generally be used only when the node is needed as an input for further work and recomputing it is prohibitively costly.

Default: NO

5.136.3.2 format

```
- (MPSImageFeatureChannelFormat) format [read], [write], [nonatomic], [assign]
```

The preferred precision for the image Default: MPSImageFeatureChannelFormatNone, meaning MPS should pick a format Typically, this is 16-bit floating-point.

5.136.3.3 handle

```
- (id<MPSHandle>) handle [read], [nonatomic], [assign]
```

MPS resource identifier See MPSHandle protocol description. Default: nil

5.136.3.4 imageAllocator

```
- (id<MPSImageAllocator>) imageAllocator [read], [write], [nonatomic], [retain]
```

Configurability for image allocation Allows you to influence how the image is allocated Default: defaultAllocator (MPSTemporaryImage)

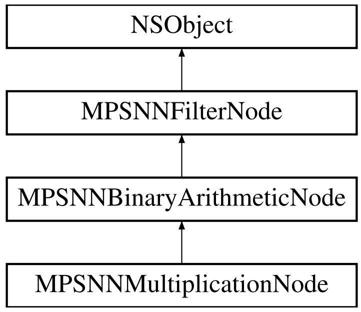
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.137 MPSNNMultiplicationNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSNNMultiplicationNode:



Additional Inherited Members

5.137.1 Detailed Description

returns elementwise product of left ∗ right

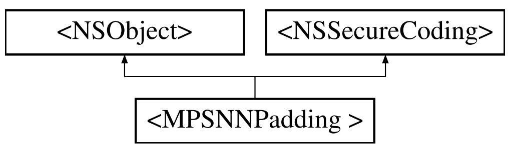
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.138 <MPSNNPadding > Protocol Reference

```
#import <MPSNeuralNetworkTypes.h>
```

Inheritance diagram for <MPSNNPadding >:



Instance Methods

- (MPSNNPaddingMethod) - paddingMethod
- (NSString ∗__nonnull) - label
- (MPSImageDescriptor ∗__nonnull) - destinationImageDescriptorForSourceImages:sourceStates:forKernel↩ :suggestedDescriptor:

5.138.1 Method Documentation

5.138.1.1 destinationImageDescriptorForSourceImages:sourceStates:forKernel:suggestedDescriptor:()

```
- (MPSImageDescriptor * __nonnull MPSNNPadding) destinationImageDescriptorForSourceImages:
            (NSArray< MPSImage * > *__nonnull) sourceImages
            sourceStates:(NSArray< MPSState * > *__nullable) sourceStates
            forKernel:(MPSKernel *__nonnull) kernel
            suggestedDescriptor:(MPSImageDescriptor *__nonnull) inDescriptor   [optional]
```

Determine padding and sizing of result images A MPSNNPaddingMethod must both return a valid MPSImageDescriptor and set the MPSKernel.offset to the correct value. This is a required feature if the MPSNNPaddingMethodCustom bit is set in the paddingMethod.

Some code that may prove helpful:

```
const int centeringPolicy = 0;  // When kernelSize is even: 0 pad bottom right. 1 pad top left.    Centers
       the kernel for even sized kernels.

typedef enum Style{
    StyleValidOnly = -1,
    StyleSame = 0,
    StyleFull = 1
}Style;

// Typical destination size in one dimension for forward filters (most filters)
static int DestSize( int sourceSize, int stride, int filterWindowSize, Style style ){
    sourceSize += style * (filterWindowSize - 1);       // adjust how many pixels we are allowed to read
    return (sourceSize + stride - 1) / stride;          // sourceSize / stride, round up
}

// Typical destination size in one dimension for reverse filters (e.g. unpooling, convolution transpose)
static int DestSizeReverse( int sourceSize, int stride, int filterWindowSize, Style style ){
    return (sourceSize-1) * stride +        // center tap for the last N-1 results. Take stride into
       account
            1 +                             // center tap for the first result
            style * (filterWindowSize-1);   // add or subtract (or ignore) the filter extent
}

// Find the MPSOffset in one dimension
static int Offset( int sourceSize, int stride, int filterWindowSize, Style style ){
    // The correction needed to adjust from position of left edge to center per MPSOffset definition
    int correction = filterWindowSize / 2;

    // exit if all we want is to start consuming pixels at the left edge of the image.
    if( 0 )
        return correction;

    // Center the area consumed in the source image:
    // Calculate the size of the destination image
    int destSize = DestSize( sourceSize, stride, filterWindowSize, style ); // use DestSizeReverse here
       instead as appropriate

    // calculate extent of pixels we need to read in source to populate the destination
    int readSize = (destSize-1) * stride + filterWindowSize;

    // calculate number of missing pixels in source
    int extraSize = readSize - sourceSize;

    // number of missing pixels on left side
    int leftExtraPixels = (extraSize + centeringPolicy) / 2;

    // account for the fact that the offset is based on the center pixel, not the left edge
    return correction - leftExtraPixels;
}
```

Parameters

sourceImages	The list of source images to be used

Parameters

sourceStates	The list of source states to be used
kernel	The MPSKernel the padding method will be applied to. Set the kernel.offset
inDescriptor	MPS will prepare a starting guess based on the padding policy (exclusive of MPSNNPaddingMethodCustom) set for the object. You should adjust the offset and image size accordingly. It is on an autoreleasepool.

Returns

The MPSImageDescriptor to use to make a MPSImage to capture the results from the filter. The MPSImage↩ Descriptor is assumed to be on an autoreleasepool. Your method must also set the kernel.offset property.

5.138.1.2 label()

```
- (NSString*__nonnull MPSNNPadding) label [optional]
```

A human readable string that describes the padding policy. Useful for verbose debugging support.

5.138.1.3 paddingMethod()

```
- (MPSNNPaddingMethod MPSNNPadding) paddingMethod [required]
```

Get the preferred padding method for the node

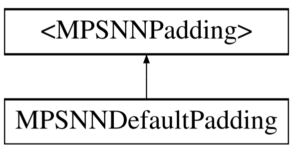
The documentation for this protocol was generated from the following file:

- MPSNeuralNetworkTypes.h

5.139 <MPSNNPadding> Protocol Reference

```
#include <MPSNeuralNetworkTypes.h>
```

Inheritance diagram for <MPSNNPadding>:



5.139.1 Detailed Description

A method to describe how MPSCNNKernels should pad images when data outside the image is needed Different (non-Apple) CNN frameworks have different policies for how to size the result of a CNN filter and what padding to add around the edges. Some filters such as pooling and convolution read from neighboring feature channel (pixel) values. Four predefined MPSPaddingMethods are available: MPSNNPaddingMethodValidOnly, MPSNNPadding↵MethodFull, MPSNNPaddingMethodSameTL, MPSNNPaddingMethodSameBR. You may also implement your own padding definition with a block that conforms to this prototype.

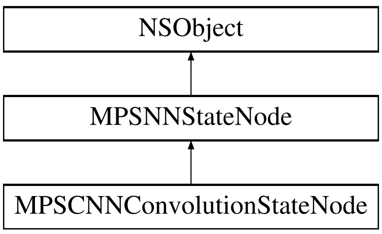
The documentation for this protocol was generated from the following file:

- MPSNeuralNetworkTypes.h

5.140 MPSNNStateNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSNNStateNode:



Instance Methods

- (nonnull instancetype) - init

Properties

- id< MPSHandle > handle

5.140.1 Detailed Description

A placeholder node denoting the position in the graph of a MPSState object Some filters need additional information about an image in order to function. For example an unpooling max filter needs to know which position the max result came from in the original pooling filter in order to select the right data for unpooling. In other cases, state may be moved into a MPSState object in order to keep the filter itself immutable. The MPSState object typically encapsulates one or more MTLResource objects.

5.140.2 Method Documentation

5.140.2.1 init()

```
- (nonnull instancetype) init
```

5.140.3 Property Documentation

5.140.3.1 handle

```
- (id<MPSHandle>) handle [read], [write], [nonatomic], [retain]
```

MPS resource identification See MPSHandle protocol reference. Default: nil

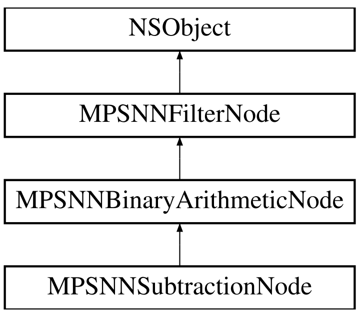
The documentation for this class was generated from the following file:

- MPSNNGraphNodes.h

5.141 MPSNNSubtractionNode Class Reference

```
#import <MPSNNGraphNodes.h>
```

Inheritance diagram for MPSNNSubtractionNode:



Additional Inherited Members

5.141.1 Detailed Description

returns elementwise difference of left - right

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

- MPSNNGraphNodes.h

5.142 MPSOffset Struct Reference

```
#include <MPSCoreTypes.h>
```

Public Attributes

- NSInteger x
- NSInteger y
- NSInteger z

5.142.1 Detailed Description

A signed coordinate with x, y and z components

5.142.2 Member Data Documentation

5.142.2.1 x

```
NSInteger MPSOffset::x
```

The horizontal component of the offset. Units: pixels

5.142.2.2 y

```
NSInteger MPSOffset::y
```

The vertical component of the offset. Units: pixels

5.142.2.3 z

```
NSInteger MPSOffset::z
```

The depth component of the offset. Units: pixels

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

- MPSCoreTypes.h

5.143 MPSOrigin Struct Reference

```
#include <MPSCoreTypes.h>
```

Public Attributes

- double x
- double y
- double z

5.143.1 Detailed Description

A position in an image

5.143.2 Member Data Documentation

5.143.2.1 x

`double MPSOrigin::x`

The x coordinate of the position

5.143.2.2 y

`double MPSOrigin::y`

The y coordinate of the position

5.143.2.3 z

`double MPSOrigin::z`

The z coordinate of the position

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

- MPSCoreTypes.h

5.144 MPSRegion Struct Reference

```
#include <MPSCoreTypes.h>
```

Public Attributes

- MPSOrigin origin
- MPSSize size

5.144.1 Detailed Description

A region of an image

5.144.2 Member Data Documentation

5.144.2.1 origin

`MPSOrigin MPSRegion::origin`

The top left corner of the region. Units: pixels

5.144.2.2 size

`MPSSize MPSRegion::size`

The size of the region. Units: pixels

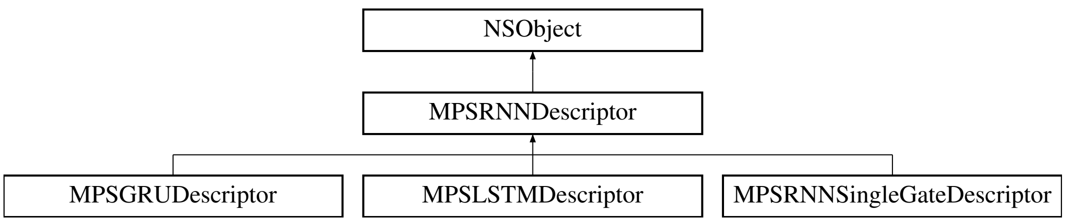
The documentation for this struct was generated from the following file:

- MPSCoreTypes.h

5.145 MPSRNNDescriptor Class Reference

`#import <MPSRNNLayer.h>`

Inheritance diagram for MPSRNNDescriptor:



Properties

- NSUInteger inputFeatureChannels
- NSUInteger outputFeatureChannels
- BOOL useLayerInputUnitTransformMode
- BOOL useFloat32Weights
- id< MPSCNNConvolutionDataSource > inputTransform
- id< MPSCNNConvolutionDataSource > outputTransform
- id< MPSCNNConvolutionDataSource > recurrentOutputTransform
- MPSRNNSequenceDirection layerSequenceDirection

5.145.1 Detailed Description

This depends on Metal.framework The MPSRNNDescriptor specifies a Recursive neural network block/layer descriptor.

5.145.2 Property Documentation

5.145.2.1 inputFeatureChannels

```
- inputFeatureChannels  [read], [write], [nonatomic], [assign]
```

The number of feature channels per pixel in the input image or number of rows in the input matrix.

5.145.2.2 inputTransform

```
- inputTransform  [read], [write], [nonatomic], [retain]
```

Transform the input of the RNN layer before any other operations. For an image based RNN this perfoms a convolution operation as specified by the data source and for a matrix based RNN it performs a matrix multiply and an affine transform - in this case the kernelWidth and kernelHeight must be equal to one in order for it to define a matrix. This applies across all the other data sources defined in the different RNN descriptors. If the normal operation of the RNN layer is: (yt, ht, ct) = f(xt,ht-1,ct-1), then this transformation replaces 'xt' with 'W∗xt + b'. If nil, no operation is performed. Defaults to nil.

5.145.2.3 layerSequenceDirection

```
- layerSequenceDirection  [read], [write], [nonatomic], [assign]
```

When the layer specified with this descriptor is used to process a sequence of inputs by calling

See also

encodeBidirectionalSequenceToCommandBuffer then this parameter defines in which direction the sequence is processed. The operation of the layer is: (yt, ht, ct) = f(xt,ht-1,ct-1) for MPSRNNSequenceDirection↩ Forward and (yt, ht, ct) = f(xt,ht+1,ct+1) for MPSRNNSequenceDirectionBackward, where xt is the output of the previous layer that encodes in the same direction as this layer, (or the input image or matrix if this is the first layer in stack with this direction).
MPSRNNImageInferenceLayer and
MPSRNNMatrixInferenceLayer.

5.145.2.4 outputFeatureChannels

```
- outputFeatureChannels  [read], [write], [nonatomic], [assign]
```

The number of feature channels per pixel in the destination image or number of rows in the destination matrix.

5.145.2.5 outputTransform

```
- outputTransform [read], [write], [nonatomic], [retain]
```

Transform the output values on the destination of the RNN layer after all other operations. For a image based RNN this perfoms a convolution operation as specified by the data source and for a matrix based RNN it performs a matrix multiply and an affine transform. If the normal operation of the RNN layer is: (yt, ht, ct) = f(xt,ht-1,ct-1), then this transformation replaces 'yt' with 'W∗yt + b'. If nil, no operation is performed on the output values. Defaults to nil.

5.145.2.6 recurrentOutputTransform

```
- recurrentOutputTransform [read], [write], [nonatomic], [retain]
```

Transform the output values on the destination of the RNN layer after all other operations. For a image based RNN this perfoms a convolution operation as specified by the data source and for a matrix based RNN it performs a matrix multiply and an affine transform. If the normal operation of the RNN layer is: (yt, ht, ct) = f(xt,ht-1,ct-1), then this transformation replaces 'ht' with 'W∗ht + b'. If nil, no operation is performed on the output values. Defaults to nil.

5.145.2.7 useFloat32Weights

```
- useFloat32Weights [read], [write], [nonatomic], [assign]
```

If YES, then MPSRNNMatrixInferenceLayer uses 32-bit floating point numbers internally for weights when computing matrix transformations. If NO, then 16-bit, half precision floating point numbers are used. Currently MPSRNNImageInferenceLayer ignores this property and the convolution operations always convert FP32 weights into FP16 for better performance. Defaults to NO.

5.145.2.8 useLayerInputUnitTransformMode

```
- useLayerInputUnitTransformMode [read], [write], [nonatomic], [assign]
```

if YES then use identity transformation for all weights (W, Wr, Wi, Wf, Wo, Wc) affecting input x_j in this layer, even if said weights are specified as nil. For example 'W_ij ∗ x_j' is replaced by 'x_j' in formulae defined in MPSBasicRNNDescriptor, but not in inputTransform. Defaults to NO.

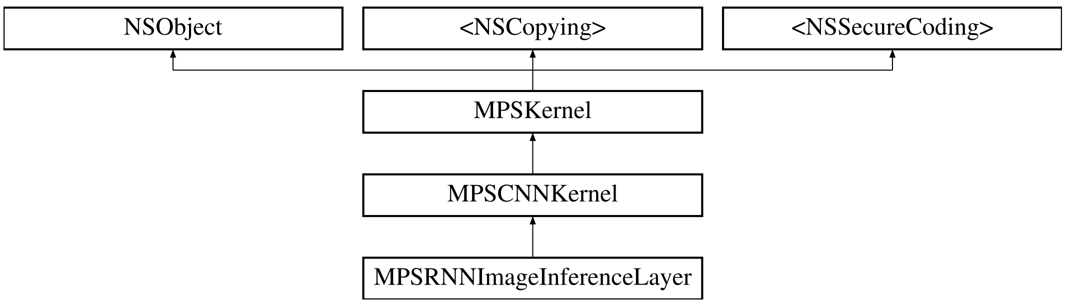
The documentation for this class was generated from the following file:

- MPSRNNLayer.h

5.146 MPSRNNImageInferenceLayer Class Reference

```
#import <MPSRNNLayer.h>
```

Inheritance diagram for MPSRNNImageInferenceLayer:



Instance Methods

- (nonnull instancetype) - initWithDevice:rnnDescriptor:
- (nonnull instancetype) - initWithDevice:rnnDescriptors:
- (nonnull instancetype) - initWithDevice:
- (void) - encodeSequenceToCommandBuffer:sourceImages:destinationImages:recurrentInputState↩ :recurrentOutputStates:
- (void) - encodeBidirectionalSequenceToCommandBuffer:sourceSequence:destinationForwardImages↩ :destinationBackwardImages:
- (nullable instancetype) - initWithCoder:device:
- (nonnull instancetype) - copyWithZone:device:

Properties

- NSUInteger inputFeatureChannels
- NSUInteger outputFeatureChannels
- NSUInteger numberOfLayers
- BOOL recurrentOutputIsTemporary
- BOOL storeAllIntermediateStates
- MPSRNNBidirectionalCombineMode bidirectionalCombineMode

Additional Inherited Members

5.146.1 Detailed Description

This depends on Metal.framework The MPSRNNImageInferenceLayer specifies a recurrent neural network layer for inference on MPSImages. Currently two types of recurrent layers are supported: ones that operate with convolutions on images: MPSRNNImageInferenceLayer and one that operates on matrices: MPSRNNMatrixInferenceLayer. The former can be often used to implement the latter by using 1x1-images, but due to image size restrictions and performance, it is advisable to use MPSRNNMatrixInferenceLayer for linear recurrent layers. A MPSRNNImage↩ InferenceLayer is initialized using a MPSRNNLayerDescriptor, which further specifies the recurrent network layer, or an array of MPSRNNLayerDescriptors, which specifies a stack of recurrent layers, that can operate in parallel a subset of the inputs in a sequence of inputs and recurrent outputs. Note that currently stacks with bidirectionally traversing encode functions do not support starting from a previous set of recurrent states, but this can be achieved quite easily by defining two separate unidirectional stacks of layers, and running the same input sequence on them separately (one forwards and one backwards) and ultimately combining the two result sequences as desired with auxiliary functions.

5.146.2 Method Documentation

5.146.2.1 copyWithZone:device:()

```
- (nonnull instancetype) copyWithZone:
            (nullable NSZone *) zone
            device:(nullable id< MTLDevice >) device
```

Make a copy of this kernel for a new device -

See also

MPSKernel

Parameters

zone	The NSZone in which to allocate the object
device	The device for the new MPSKernel. If nil, then use self.device.

Returns

a pointer to a copy of this MPSKernel. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSKernel.

5.146.2.2 encodeBidirectionalSequenceToCommandBuffer:sourceSequence:destinationForwardImages:destinationBackward↩ Images:()

```
- (void) encodeBidirectionalSequenceToCommandBuffer:
            (nonnull id< MTLCommandBuffer >) commandBuffer
            sourceSequence:(NSArray< MPSImage * > *__nonnull) sourceSequence
            destinationForwardImages:(NSArray< MPSImage * > *__nonnull) destinationForward↩
Images
            destinationBackwardImages:(NSArray< MPSImage * > *__nullable) destinationBackward↩
Images
```

Encode an MPSRNNImageInferenceLayer kernel stack for an input image sequences into a command buffer bidirectionally. The operation proceeds as follows: The first source image x0 is passed through all forward traversing layers in the stack, ie. those that were initialized with MPSRNNSequenceDirectionForward, recurrent input is assumed zero. This produces forward output yf0 and recurrent states hf00, hf01, hf02, ... hf0n, one for each forward layer. Then x1 is passed to forward layers together with recurrent state hf00, hf01, ..., hf0n, which produces yf1, and hf10,... This procedure is iterated until the last image in the input sequence x_(N-1), which produces forward output yf(N-1). The backwards layers iterate the same sequence backwards, starting from input x_(N-1) (recurrent state zero), that produces yb(N-1) and recurrent output hb(N-1)0, hf(N-1)1, ... hb(N-1)m, one for each backwards traversing layer. Then the backwards layers handle input x_(N-2) using recurrent state hb(N-1)0, ..., et cetera, until the first image of the sequence is computed, producing output yb0. The result of the operation is either pair of sequences ({yf0, yf1, ... , yf(N-1)}, {yb0, yb1, ... , yb(N-1)}) or a combined sequence, {(yf0 + yb0), ... , (yf(N-1) + yb(N-1)) }, where '+' stands either for sum, or concatenation along feature channels, as specified by bidirectional↩ CombineMode.

Parameters

commandBuffer	A valid MTLCommandBuffer to receive the encoded filter
sourceSequence	An array of valid MPSImage objects containing the source image sequence (x0, x1, ... x_n-1).
destinationForwardImages	An array of valid MPSImages to be overwritten by result from forward input images. If bidirectionalCombineMode is either MPSRNNBidirectionalCombineModeAdd or MPSRNNBidirectionalCombineModeConcatenate, then will contain the combined results. destinationForwardImage may not alias with any of the source images.
destinationBackwardImages	If bidirectionalCombineMode is MPSRNNBidirectionalCombineModeNone, then must be a valid MPSImage that will be overwritten by result from backward input image. Otherwise this parameter is ignored and can be nil. destinationBackwardImages may not alias to any of the source images.

5.146.2.3 encodeSequenceToCommandBuffer:sourceImages:destinationImages:recurrentInputState:recurrentOutputStates:()

```
- (void) encodeSequenceToCommandBuffer:
            (nonnull id< MTLCommandBuffer >) commandBuffer
            sourceImages:(NSArray< MPSImage * > *__nonnull) sourceImages
            destinationImages:(NSArray< MPSImage * > *__nonnull) destinationImages
            recurrentInputState:(MPSRNNRecurrentImageState *__nullable) recurrentInputState
            recurrentOutputStates:(NSMutableArray< MPSRNNRecurrentImageState * > *__nullable)
recurrentOutputStates
```

Encode an MPSRNNImageInferenceLayer kernel (stack) for a sequence of inputs into a command buffer. Note that when encoding using this function the

See also

layerSequenceDirection is ignored and the layer stack operates as if all layers were forward feeding layers. In order to run bidirectional sequences use encodeBidirectionalSequenceToCommandBuffer:sourceSequence: or alternatively run two layer stacks and combine results at the end using utility functions.

Parameters

commandBuffer	A valid MTLCommandBuffer to receive the encoded filter
sourceImages	An array of valid MPSImage objects containing the sequence of source images.
destinationImages	An array valid MPSImages to be overwritten by result image sequence. destinationImages may not alias sourceImages.
recurrentInputState	An optional state containing the output images and memory cells (for LSTMs) of the layer obtained from the previous input images in a sequence of inputs. Has to be the output of a previous call to this function or nil (assumed zero). Note: can be one of the states returned in recurrentOutputStates.
recurrentOutputStates	An optional array that will contain the recurrent output states. If nil then the recurrent output state is discarded. If storeAllIntermediateStates is YES, then all intermediate states of the sequence are returned in the array, the first one corresponding to the first input in the sequence, otherwise only the last recurrent output state is returned. If recurrentOutputIsTemporary is YES and then all returned recurrent states will be temporary.

See also

MPSState:isTemporary. Example: In order to get a new state one can do the following:

```
MPSRNNRecurrentImageState* recurrent0 = nil;
[filter encodeToCommandBuffer: cmdBuf
                  sourceImage: source0
             destinationImage: destination0
          recurrentInputState: nil
         recurrentOutputState: &recurrent0];
```

Then use it for the next input in sequence:

```
[filter encodeToCommandBuffer: cmdBuf
                  sourceImage: source1
             destinationImage: destination1
          recurrentInputState: recurrent0
         recurrentOutputState: &recurrent0];
```

And discard recurrent output of the third input:

```
[filter encodeToCommandBuffer: cmdBuf
                  sourceImage: source2
             destinationImage: destination2
          recurrentInputState: recurrent0
         recurrentOutputState: nil];
```

5.146.2.4 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability See MPSKernel::initWithCoder.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSRNNImageInferenceLayer
device	The MTLDevice on which to make the MPSRNNImageInferenceLayer

Returns

A new MPSRNNImageInferenceLayer object, or nil if failure.

Reimplemented from MPSCNNKernel.

5.146.2.5 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

A pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSCNNKernel.

5.146.2.6 initWithDevice:rnnDescriptor:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            rnnDescriptor:(nonnull const MPSRNNDescriptor *) rnnDescriptor
```

Initializes a convolutional RNN kernel

Parameters

device	The MTLDevice on which this MPSRNNImageLayer filter will be used
rnnDescriptor	The descriptor that defines the RNN layer

Returns

A valid MPSRNNImageInferenceLayer object or nil, if failure.

5.146.2.7 initWithDevice:rnnDescriptors:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            rnnDescriptors:(NSArray< const MPSRNNDescriptor * > *__nonnull) rnnDescriptors
```

Initializes a kernel that implements a stack of convolutional RNN layers

Parameters

device	The MTLDevice on which this MPSRNNImageLayer filter will be used
rnnDescriptors	An array of RNN descriptors that defines a stack of RNN layers, starting at index zero. The number of layers in stack is the number of entries in the array. All entries in the array must be valid MPSRNNDescriptors.

Returns

A valid MPSRNNImageInferenceLayer object or nil, if failure.

5.146.3 Property Documentation

5.146.3.1 bidirectionalCombineMode

```
- bidirectionalCombineMode  [read], [write], [nonatomic], [assign]
```

Defines how to combine the output-results, when encoding bidirectional layers using encodeBidirectional↩SequenceToCommandBuffer. Defaults to MPSRNNBidirectionalCombineModeNone.

5.146.3.2 inputFeatureChannels

```
- inputFeatureChannels [read], [nonatomic], [assign]
```

The number of feature channels per pixel in the input image.

5.146.3.3 numberOfLayers

```
- numberOfLayers [read], [nonatomic], [assign]
```

Number of layers in the filter-stack. This will be one when using initWithDevice:rnnDescriptor to initialize this filter and the number of entries in the array 'rnnDescriptors' when initializing this filter with initWithDevice:rnnDescriptors.

5.146.3.4 outputFeatureChannels

```
- outputFeatureChannels [read], [nonatomic], [assign]
```

The number of feature channels per pixel in the output image.

5.146.3.5 recurrentOutputIsTemporary

```
- recurrentOutputIsTemporary [read], [write], [nonatomic], [assign]
```

How output states from encodeSequenceToCommandBuffer are constructed. Defaults to NO. For reference

See also

MPSState.

5.146.3.6 storeAllIntermediateStates

```
- storeAllIntermediateStates [read], [write], [nonatomic], [assign]
```

If YES then calls to encodeSequenceToCommandBuffer return every recurrent state in the array: recurrentOutput←States. Defaults to NO.

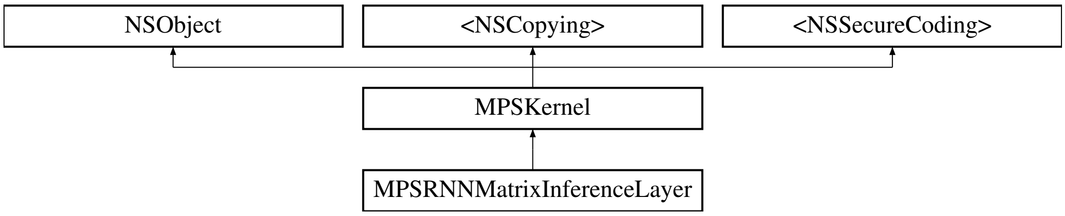
The documentation for this class was generated from the following file:

- MPSRNNLayer.h

5.147 MPSRNNMatrixInferenceLayer Class Reference

```
#import <MPSRNNLayer.h>
```

Inheritance diagram for MPSRNNMatrixInferenceLayer:



Instance Methods

- (nonnull instancetype) - initWithDevice:rnnDescriptor:
- (nonnull instancetype) - initWithDevice:rnnDescriptors:
- (nonnull instancetype) - initWithDevice:
- (void) - encodeSequenceToCommandBuffer:sourceMatrices:destinationMatrices:recurrentInputState↩ :recurrentOutputStates:
- (void) - encodeBidirectionalSequenceToCommandBuffer:sourceSequence:destinationForwardMatrices↩ :destinationBackwardMatrices:
- (nullable instancetype) - initWithCoder:device:
- (nonnull instancetype) - copyWithZone:device:

Properties

- NSUInteger inputFeatureChannels
- NSUInteger outputFeatureChannels
- NSUInteger numberOfLayers
- BOOL recurrentOutputIsTemporary
- BOOL storeAllIntermediateStates
- MPSRNNBidirectionalCombineMode bidirectionalCombineMode

Additional Inherited Members

5.147.1 Detailed Description

This depends on Metal.framework The MPSRNNMatrixInferenceLayer specifies a recurrent neural network layer for inference on MPSMatrices. Currently two types of recurrent layers are supported: ones that operate with convolutions on images: MPSRNNImageInferenceLayer and one that operates on matrices: MPSRNNMatrix↩ InferenceLayer. The former can be often used to implement the latter by using 1x1-matrices, but due to image size restrictions and performance, it is advisable to use MPSRNNMatrixInferenceLayer for linear recurrent layers. A MPSRNNMatrixInferenceLayer is initialized using a MPSRNNLayerDescriptor, which further specifies the recurrent network layer, or an array of MPSRNNLayerDescriptors, which specifies a stack of recurrent layers, that can operate in parallel a subset of the inputs in a sequence of inputs and recurrent outputs. Note that currently stacks with bidirectionally traversing encode functions do not support starting from a previous set of recurrent states, but this can be achieved quite easily by defining two separate unidirectional stacks of layers, and running the same input sequence on them separately (one forwards and one backwards) and ultimately combining the two result sequences as desired with auxiliary functions. The input and output vectors in encode calls are stored as rows of the input and output matrices and currently MPSRNNMatrixInferenceLayer supports only matrices with number of rows equal to one. The mathematical operation then is strictly speaking $y^T = W x^T \iff y = x W^T$ in the linear transformations of MPSRNNSingleGateDescriptor, MPSLSTMDescriptor and MPSGRUDescriptor.

5.147.2 Method Documentation

5.147.2.1 copyWithZone:device:()

```
- (nonnull instancetype) copyWithZone:
            (nullable NSZone *) zone
            device:(nullable id< MTLDevice >) device
```

Make a copy of this kernel for a new device -

See also

MPSKernel

Parameters

zone	The NSZone in which to allocate the object
device	The device for the new MPSKernel. If nil, then use self.device.

Returns

a pointer to a copy of this MPSKernel. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSKernel.

5.147.2.2 encodeBidirectionalSequenceToCommandBuffer:sourceSequence:destinationForwardMatrices:destinationBackward↩ Matrices:()

```
- (void) encodeBidirectionalSequenceToCommandBuffer:
            (nonnull id< MTLCommandBuffer >) commandBuffer
            sourceSequence:(NSArray< MPSMatrix * > *__nonnull) sourceSequence
            destinationForwardMatrices:(NSArray< MPSMatrix * > *__nonnull) destination↩
ForwardMatrices
            destinationBackwardMatrices:(NSArray< MPSMatrix * > *__nullable) destination↩
BackwardMatrices
```

Encode an MPSRNNMatrixInferenceLayer kernel stack for an input matrix sequences into a command buffer bidirectionally. The operation proceeds as follows: The first source matrix x0 is passed through all forward traversing layers in the stack, ie. those that were initialized with MPSRNNSequenceDirectionForward, recurrent input is assumed zero. This produces forward output yf0 and recurrent states hf00, hf01, hf02, ... hf0n, one for each forward layer in the stack. Then x1 is passed to forward layers together with recurrent state hf00, hf01, ..., hf0n, which produces yf1, and hf10,... This procedure is iterated until the last matrix in the input sequence x_(N-1), which produces forward output yf(N-1). The backwards layers iterate the same sequence backwards, starting from input x_(N-1) (recurrent state zero), that produces yb(N-1) and recurrent output hb(N-1)0, hf(N-1)1, ... hb(N-1)m, one for each backwards traversing layer. Then the backwards layers handle input x_(N-2) using recurrent state hb(N-1)0, ..., et cetera, until the first matrix of the sequence is computed, producing output yb0. The result of the operation is either pair of sequences ({yf0, yf1, ... , yf(N-1)}, {yb0, yb1, ... , yb(N-1)}) or a combined sequence, {(yf0 + yb0), ... , (yf(N-1) + yb(N-1)) }, where '+' stands either for sum, or concatenation along feature channels, as specified by bidirectionalCombineMode.

Parameters

commandBuffer	A valid MTLCommandBuffer to receive the encoded filter
sourceSequence	An array of valid MPSMatrix objects containing the source matrix sequence (x0, x1, ... x_n-1).
destinationForwardMatrices	An array of valid MPSMatrices to be overwritten by result from forward input matrices. If bidirectionalCombineMode is either MPSRNNBidirectionalCombineModeAdd or MPSRNNBidirectionalCombineModeConcatenate, then will contain the combined results. destinationForwardMatrix may not alias with any of the source matrices.
destinationBackwardMatrices	If bidirectionalCombineMode is MPSRNNBidirectionalCombineModeNone, then must be an array of valid MPSMatrices that will be overwritten by result from backward input matrices. Otherwise this parameter is ignored and can be nil. destinationBackwardMatrices may not alias to any of the source matrices.

5.147.2.3 encodeSequenceToCommandBuffer:sourceMatrices:destinationMatrices:recurrentInputState:recurrentOutput←States:()

```
- (void) encodeSequenceToCommandBuffer:
            (nonnull id< MTLCommandBuffer >) commandBuffer
            sourceMatrices:(NSArray< MPSMatrix * > *__nonnull) sourceMatrices
            destinationMatrices:(NSArray< MPSMatrix * > *__nonnull) destinationMatrices
            recurrentInputState:(MPSRNNRecurrentMatrixState *__nullable) recurrentInputState
            recurrentOutputStates:(NSMutableArray< MPSRNNRecurrentMatrixState * > *__nullable)
recurrentOutputStates
```

Encode an MPSRNNMatrixInferenceLayer kernel (stack) for a sequence of inputs into a command buffer. Note that when encoding using this function the

See also

layerSequenceDirection is ignored and the layer stack operates as if all layers were forward feeding layers. In order to run bidirectional sequences use encodeBidirectionalSequenceToCommandBuffer:sourceSequence: or alternatively run two layer stacks and combine results at the end using utility functions.

Parameters

commandBuffer	A valid MTLCommandBuffer to receive the encoded filter
sourceMatrices	An array of valid MPSMatrix objects containing the sequence of source matrices.
destinationMatrices	An array valid MPSMatrices to be overwritten by result matrix sequence. destinationMatrices may not alias sourceMatrices.
recurrentInputState	An optional state containing the output matrices and memory cells (for LSTMs) of the layer obtained from the previous input matrices in a sequence of inputs. Has to be the output of a previous call to this function or nil (assumed zero). Note: can be one of the states returned in intermediateRecurrentStates.
recurrentOutputStates	An optional array that will contain the recurrent output states. If nil then the recurrent output state is discarded. If storeAllIntermediateStates is YES, then all intermediate states of the sequence are returned in the array, the first one corresponding to the first input in the sequence, otherwise only the last recurrent output state is returned. If recurrentOutputIsTemporary is YES and then all returned recurrent states will be temporary.

See also

MPSState:isTemporary. Example: In order to get a new state one can do the following:

```
MPSRNNRecurrentMatrixState* recurrent0 = nil;
[filter encodeToCommandBuffer: cmdBuf
                sourceMatrix: source0
           destinationMatrix: destination0
         recurrentInputState: nil
        recurrentOutputState: &recurrent0];
```

Then use it for the next input in sequence:

```
[filter encodeToCommandBuffer: cmdBuf
                sourceMatrix: source1
           destinationMatrix: destination1
         recurrentInputState: recurrent0
        recurrentOutputState: &recurrent0];
```

And discard recurrent output of the third input:

```
[filter encodeToCommandBuffer: cmdBuf
                sourceMatrix: source2
           destinationMatrix: destination2
         recurrentInputState: recurrent0
        recurrentOutputState: nil];
```

5.147.2.4 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability See MPSKernel::initWithCoder.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSRNNMatrixInferenceLayer
device	The MTLDevice on which to make the MPSRNNMatrixInferenceLayer

Returns

A new MPSRNNMatrixInferenceLayer object, or nil if failure.

Reimplemented from MPSKernel.

5.147.2.5 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

a pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSKernel.

5.147.2.6 initWithDevice:rnnDescriptor:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            rnnDescriptor:(nonnull const MPSRNNDescriptor *) rnnDescriptor
```

Initializes a linear (fully connected) RNN kernel

Parameters

device	The MTLDevice on which this MPSRNNMatrixLayer filter will be used
rnnDescriptor	The descriptor that defines the RNN layer

Returns

A valid MPSRNNMatrixInferenceLayer object or nil, if failure.

5.147.2.7 initWithDevice:rnnDescriptors:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            rnnDescriptors:(NSArray< const MPSRNNDescriptor * > *__nonnull) rnnDescriptors
```

Initializes a kernel that implements a stack of linear (fully connected) RNN layers

Parameters

device	The MTLDevice on which this MPSRNNMatrixLayer filter will be used
rnnDescriptors	An array of RNN descriptors that defines a stack of RNN layers, starting at index zero. The number of layers in stack is the number of entries in the array. All entries in the array must be valid MPSRNNDescriptors.

Returns

A valid MPSRNNMatrixInferenceLayer object or nil, if failure.

5.147.3 Property Documentation

5.147.3.1 bidirectionalCombineMode

```
- bidirectionalCombineMode  [read], [write], [nonatomic], [assign]
```

Defines how to combine the output-results, when encoding bidirectional layers using encodeBidirectional←SequenceToCommandBuffer. Defaults to MPSRNNBidirectionalCombineModeNone.

5.147.3.2 inputFeatureChannels

```
- inputFeatureChannels  [read], [nonatomic], [assign]
```

The number of feature channels input vector/matrix.

5.147.3.3 numberOfLayers

```
- numberOfLayers  [read], [nonatomic], [assign]
```

Number of layers in the filter-stack. This will be one when using initWithDevice:rnnDescriptor to initialize this filter and the number of entries in the array 'rnnDescriptors' when initializing this filter with initWithDevice:rnnDescriptors.

5.147.3.4 outputFeatureChannels

```
- outputFeatureChannels  [read], [nonatomic], [assign]
```

The number of feature channels in the output vector/matrix.

5.147.3.5 recurrentOutputIsTemporary

```
- recurrentOutputIsTemporary  [read], [write], [nonatomic], [assign]
```

How output states from encodeSequenceToCommandBuffer are constructed. Defaults to NO. For reference

See also

MPSState.

5.147.3.6 storeAllIntermediateStates

```
- storeAllIntermediateStates  [read], [write], [nonatomic], [assign]
```

If YES then calls to encodeSequenceToCommandBuffer return every recurrent state in the array: recurrentOutput↩States. Defaults to NO.

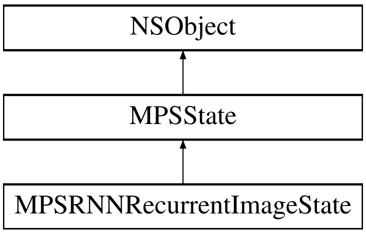
The documentation for this class was generated from the following file:

- MPSRNNLayer.h

5.148 MPSRNNRecurrentImageState Class Reference

```
#import <MPSRNNLayer.h>
```

Inheritance diagram for MPSRNNRecurrentImageState:



Instance Methods

- (nullable MPSImage ∗) - getRecurrentOutputImageForLayerIndex:
- (nullable MPSImage ∗) - getMemoryCellImageForLayerIndex:

Additional Inherited Members

5.148.1 Detailed Description

This depends on Metal.framework This class holds all the data that is passed from one sequence iteration of the image-based RNN layer (stack) to the next.

5.148.2 Method Documentation

5.148.2.1 getMemoryCellImageForLayerIndex:()

```
- (nullable MPSImage*) getMemoryCellImageForLayerIndex:
            (NSUInteger) layerIndex
```

Access the stored memory cell image data (if present).

Parameters

layerIndex	Index of the layer whose to get - belongs to { 0, 1,...,

See also

numberOfLayers - 1 }

Returns

For valid layerIndex the memory cell image data, otherwise nil.

5.148.2.2 getRecurrentOutputImageForLayerIndex:()

```
- (nullable MPSImage*) getRecurrentOutputImageForLayerIndex:
              (NSUInteger) layerIndex
```

Access the stored recurrent image data.

Parameters

layerIndex	Index of the layer whose to get - belongs to { 0, 1,...,

See also

numberOfLayers - 1 }

Returns

For valid layerIndex the recurrent output image data, otherwise nil.

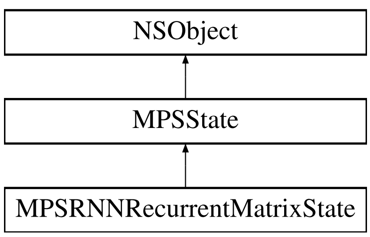
The documentation for this class was generated from the following file:

- MPSRNNLayer.h

5.149 MPSRNNRecurrentMatrixState Class Reference

```
#import <MPSRNNLayer.h>
```

Inheritance diagram for MPSRNNRecurrentMatrixState:



Instance Methods

- (nullable MPSMatrix ∗) - getRecurrentOutputMatrixForLayerIndex:
- (nullable MPSMatrix ∗) - getMemoryCellMatrixForLayerIndex:

Additional Inherited Members

5.149.1 Detailed Description

This depends on Metal.framework This class holds all the data that is passed from one sequence iteration of the matrix-based RNN layer to the next.

5.149.2 Method Documentation

5.149.2.1 getMemoryCellMatrixForLayerIndex:()

```
- (nullable MPSMatrix*) getMemoryCellMatrixForLayerIndex:
            (NSUInteger) layerIndex
```

Access the stored memory cell matrix data (if present).

Parameters

layerIndex	Index of the layer whose to get - belongs to { 0, 1,...,

See also

numberOfLayers - 1 }

Returns

For valid layerIndex the memory cell image matrix, otherwise nil.

5.149.2.2 getRecurrentOutputMatrixForLayerIndex:()

```
- (nullable MPSMatrix*) getRecurrentOutputMatrixForLayerIndex:
            (NSUInteger) layerIndex
```

Access the stored recurrent matrix data.

Parameters

layerIndex	Index of the layer whose to get - belongs to { 0, 1,...,

See also

numberOfLayers - 1 }

Returns

For valid layerIndex the recurrent output matrix data, otherwise nil.

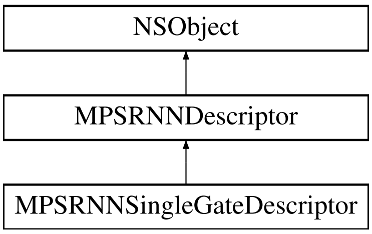
The documentation for this class was generated from the following file:

- MPSRNNLayer.h

5.150 MPSRNNSingleGateDescriptor Class Reference

```
#import <MPSRNNLayer.h>
```

Inheritance diagram for MPSRNNSingleGateDescriptor:



Class Methods

- (nonnull instancetype) + createRNNSingleGateDescriptorWithInputFeatureChannels:outputFeature↩Channels:

Properties

- id< MPSCNNConvolutionDataSource > inputWeights
- id< MPSCNNConvolutionDataSource > recurrentWeights

5.150.1 Detailed Description

This depends on Metal.framework The MPSRNNSingleGateDescriptor specifies a simple recurrent block/layer descriptor. The RNN layer initialized with a MPSRNNSingleGateDescriptor transforms the input data (image or matrix), and previous output with a set of filters, each producing one feature map in the new output data. The user may provide the RNN unit a single input or a sequence of inputs.

```
Description of operation:
```

Let x_j be the input data (at time index t of sequence, j index containing quadruplet: batch index, x,y and feature index (x=y=0 for matrices)). Let h0_j be the recurrent input (previous output) data from previous time step (at time index t-1 of sequence). Let h1_i be the output data produced at this time step.

Let W_ij, U_ij be the weights for input and recurrent input data respectively Let b_i be a bias term

Let gi(x) be a neuron activation function

Then the new output image h1_i data is computed as follows:

```
h1_i = gi( W_ij * x_j + U_ij * h0_j  + b_i )
```

The '*' stands for convolution (see MPSRNNImageInferenceLayer) or matrix-vector/matrix multiplication (see MPSRNNMatrixInferenceLayer). Summation is over index j (except for the batch index), but there is no summation over repeated index i - the output index. Note that for validity all intermediate images have to be of same size and the U matrix has to be square (ie. outputFeatureChannels == inputFeatureChannels in those). Also the bias terms are scalars wrt. spatial dimensions.

5.150.2 Method Documentation

5.150.2.1 createRNNSingleGateDescriptorWithInputFeatureChannels:outputFeatureChannels:()

```
+ (nonnull instancetype) createRNNSingleGateDescriptorWithInputFeatureChannels:
            (NSUInteger) inputFeatureChannels
            outputFeatureChannels:(NSUInteger) outputFeatureChannels
```

Creates a MPSRNNSingleGateDescriptor

Parameters

inputFeatureChannels	The number of feature channels in the input image/matrix. Must be >= 1.
outputFeatureChannels	The number of feature channels in the output image/matrix. Must be >= 1.

Returns

A valid MPSRNNSingleGateDescriptor object or nil, if failure.

5.150.3 Property Documentation

5.150.3.1 inputWeights

```
- inputWeights [read], [write], [nonatomic], [retain]
```

Contains weights 'W_ij', bias 'b_i' and neuron 'gi' from the simple RNN layer formula. If nil then assumed zero weights, bias and no neuron (identity mapping). Defaults to nil.

5.150.3.2 recurrentWeights

```
- recurrentWeights [read], [write], [nonatomic], [retain]
```

Contains weights 'U_ij' from the simple RNN layer formula. If nil then assumed zero weights. Defaults to nil.

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

- MPSRNNLayer.h

5.151 MPSScaleTransform Struct Reference

```
#include <MPSImageTypes.h>
```

Public Attributes

- double scaleX
- double scaleY
- double translateX
- double translateY

5.151.1 Detailed Description

Transform matrix for explict control over resampling in MPSImageLanczosScale. The MPSScaleTransform is equivalent to:

```
(CGAffineTransform) {
     .a = scaleX,         .b = 0,
     .c = 0,              .d = scaleY,
     .tx = translateX,    .ty = translateY
 }
```

5.151.2 Member Data Documentation

5.151.2.1 scaleX

```
double MPSScaleTransform::scaleX
```

horizontal scaling factor

5.151.2.2 scaleY

```
double MPSScaleTransform::scaleY
```

vertical scaling factor

5.151.2.3 translateX

```
double MPSScaleTransform::translateX
```

horizontal translation

5.151.2.4 translateY

```
double MPSScaleTransform::translateY
```

vertical translation

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

- MPSImageTypes.h

5.152 MPSSize Struct Reference

```
#include <MPSCoreTypes.h>
```

Public Attributes

- double width
- double height
- double depth

5.152.1 Detailed Description

A size of a region in an image

5.152.2 Member Data Documentation

5.152.2.1 depth

```
double MPSSize::depth
```

The depth of the region

5.152.2.2 height

```
double MPSSize::height
```

The height of the region

5.152.2.3 width

```
double MPSSize::width
```

The width of the region

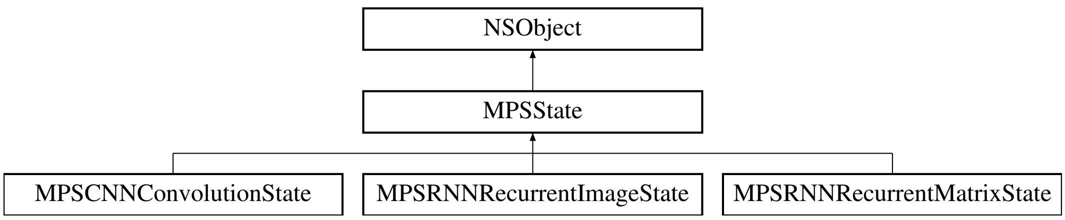
The documentation for this struct was generated from the following file:

- MPSCoreTypes.h

5.153 MPSState Class Reference

```
#import <MPSState.h>
```

Inheritance diagram for MPSState:



Instance Methods

- (nullable instancetype) - init

Properties

- NSUInteger readCount
- BOOL isTemporary
- NSString ∗ label

5.153.1 Detailed Description

This depends on Metal Framework An opaque data container for large storage in MPS CNN filters Some MPS CNN kernels produce additional information beyond a MPSImage. These may be pooling indices where the result came from, convolution weights, or other information not contained in the usual MPSImage result from a MPSCNNKernel. A MPSState object typically contains one or more expensive MTLResources such as textures or buffers to store this information. It provides a base class with interfaces for managing this storage. Child classes may add additional functionality specific to their contents.

Some MPSState objects are temporary. Temporary state objects, like MPSTemporaryImages and Matrices, are for very short lived storage, perhaps just a few lines of code within the scope of a single MTLCommandBuffer. They are very efficient for storage, as several temporary objects can share the same memory over the course of a M↩ TLCommandBuffer. This can improve both memory usage and time spent in the kernel wiring down memory and such. You may find that some large CNN tasks can not be computed without them, as non-temporary storage would simply take up too much memory.

In exchange, the lifetime of the underlying storage in temporary MPSState objects needs to be carefully managed. ARC often waits until the end of scope to release objects. Temporary storage often needs to be released sooner than that. Consequently the lifetime of the data in the underlying MTLResources is managed by a readCount property. Each time a MPSCNNKernel reads a temporary MPSState object the readCount is automatically decremented. When it reaches zero, the underlying storage is recycled for use by other MPS temporary objects, and the data is becomes undefined. If you need to consume the data multiple times, you should set the readCount to a larger number to prevent the data from becomming undefined. You may set the readCount to 0 yourself to return the storage to MPS, if for any reason, you realize that the MPSState object will no longer be used.

The contents of a temporary MPSState object are only valid from creation to the time the readCount reaches 0. The data is only valid for the MTLCommandBuffer on which it was created. Non-temporary MPSState objects are valid on any MTLCommandBuffer on the same device until they are released.

5.153.2 Method Documentation

5.153.2.1 init()

```
- (nullable instancetype) init
```

5.153.3 Property Documentation

5.153.3.1 isTemporary

```
- (BOOL) isTemporary [read], [nonatomic], [assign]
```

5.153.3.2 label

```
- label  [read], [write], [atomic], [copy]
```

A string to help identify this object.

5.153.3.3 readCount

```
- (NSUInteger) readCount  [read], [write], [nonatomic], [assign]
```

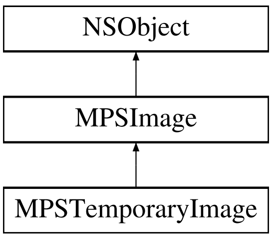
The documentation for this class was generated from the following file:

- MPSState.h

5.154 MPSTemporaryImage Class Reference

```
#import <MPSImage.h>
```

Inheritance diagram for MPSTemporaryImage:



Instance Methods

- (nonnull instancetype) - initWithTexture:featureChannels:
- (nonnull instancetype) - initWithDevice:imageDescriptor:

Class Methods

- (nonnull id< MPSImageAllocator >) + defaultAllocator
- (nonnull instancetype) + temporaryImageWithCommandBuffer:imageDescriptor:
- (nonnull instancetype) + temporaryImageWithCommandBuffer:textureDescriptor:
- (void) + prefetchStorageWithCommandBuffer:imageDescriptorList:

Properties

- NSUInteger readCount

5.154.1 Detailed Description

MPSImage MPSTemporaryImages are for MPSImages with short lifetimes.

What is temporary memory? It is memory, plain and simple. Analogy: If we use an app as an analogy for a command buffer, then "Regular memory" (such as what backs a MPSImage or the typical MTLTexture) would be memory that you allocate at launch and never free. Temporary memory would be memory that you free when you are done with it so it can be used for something else as needed later in your app. You /could/ write your app to allocate everything you will ever need up front, but this is very inefficient and quite frankly a pain to plan out in advance. You don't do it for your app, so why would you do it for your command buffers?

Welcome to the 1970's! We have added a heap.

Unsurprisingly, MPSTemporaryImages can provide for profound reduction in the the amount of memory used by your application. Like malloc, MPS maintains a heap of memory usable in a command buffer. Over the lifetime of a command buffer, the same piece of memory may be reused many times. This means that each time the same meory is reused, it aliases with previous uses. If we aren't careful, we might find that needed data is overwritten by successive allocations. However, this is no different than accessing freed memory only to discover it doesn't contain what you thought it did anymore, so you should be able to keep out of trouble by following a few simple rules, like with malloc.

To this end, we added some restrictions to help you out and get a bit more performance. Some comments are appended in parentheses below to extend the analogy of command buffer = program:

- The textures are MTLStorageModePrivate. You can not, for example, use [MTLTexture getBytes...] or [MTL↩ Texture replaceRegion...] with them. MPSTemporaryImages are strictly read and written by the GPU. (There is protected memory to prevent other processes from overwriting your heap.)
- The temporary image may be used only on a single MTLCommandBuffer. This limits the chronology to a single linear time stream. (The heap is specific to just one command buffer. There are no mutexes to coordinate timing of simultaneous access by multiple GPUs. Nor are we likely to like them if there were. So, we disallow it.)
- The readCount property must be managed correctly. Please see the description of the readCount property for full details. (The readCount is a reference count for the block of memory that holds your data. The usual undefined behaviors apply to reading data that has been released. We assert when we can to prevent that from happening accidentally, just as a program might segfault. The readCount counts procedural users of the object – MPSKernel.encode... calls that read the MPSTemporaryImage. As each reads from it, the read↩ Count is automatically decremented. The texture data will be freed in typical usage at the right time as the readCount reaches zero, typically with little user involvement other than to set the readCount up front. We did examine using the main MPSTemporaryImage reference count for this instead so that ARC would do work for you automatically. Alas, ARC destroys things at end of scope rather than promptly, sometimes resulting in greatly increased memory usage. These allocations are large! So, we use this method instead.)

Since MPSTemporaryImages can only be used with a single MTLCommandBuffer, and can not be used off the GPU, they generally should not be kept around past the completion of the MTLCommandBuffer. The lifetime of MPSTemporaryImages is expected to be typically extremely short, perhaps only a few lines of code. Like malloc, it is intended to be fairly cheap to make MPSTemporaryImages and throw them away. Please do so.

To keep the lifetime of the underlying texture allocation as short as possible, the underlying texture is not allocated until the first time the MPSTemporaryImage is used by a MPSCNNKernel or the .texture property is read. The readCount property serves to limit the lifetime on the other end.

You may use the MPSTemporaryImage.texture with MPSUnaryImageKernel -encode... methods, iff feature↩ Channels <= 4 and the MTLTexture conforms to requirements of that MPSKernel. There is no locking mechanism provided to prevent a MTLTexture returned from the .texture property from becoming invalid when the readCount reaches 0.

MPSTemporaryImages can otherwise be used wherever MPSImages are used.

5.154.2 Method Documentation

5.154.2.1 defaultAllocator()

```
+ (nonnull id <MPSImageAllocator>) defaultAllocator
```

Get a well known MPSImageAllocator that makes MPSTemporaryImages

Reimplemented from MPSImage.

5.154.2.2 initWithDevice:imageDescriptor:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
            imageDescriptor:(const MPSImageDescriptor *__nonnull) imageDescriptor
```

Unavailable. Use itemporaryImageForCommandBuffer:textureDescriptor: instead.

Reimplemented from MPSImage.

5.154.2.3 initWithTexture:featureChannels:()

```
- (nonnull instancetype) initWithTexture:
            (nonnull id< MTLTexture >) texture
            featureChannels:(NSUInteger) featureChannels
```

Unavailable. Use temporaryImageForCommandBuffer:textureDescriptor: or -temporaryImageForCommandBuffer:imageDescriptor: instead.

Reimplemented from MPSImage.

5.154.2.4 prefetchStorageWithCommandBuffer:imageDescriptorList:()

```
+ (void) prefetchStorageWithCommandBuffer:
            (nonnull id< MTLCommandBuffer >) commandBuffer
            imageDescriptorList:(NSArray< MPSImageDescriptor * > *__nonnull) descriptorList
```

Help MPS decide which allocations to make ahead of time The texture cache that underlies the MPSTemporaryImage can automatically allocate new storage as needed as you create new temporary images. However, sometimes a more global view of what you plan to make is useful for maximizing memory reuse to get the most efficient operation. This class method hints to the cache what the list of images will be.

It is never necessary to call this method. It is purely a performance and memory optimization.

Parameters

commandBuffer	The command buffer on which the MPSTemporaryImages will be used
descriptorList	A NSArray of MPSImageDescriptors, indicating images that will be created

5.154.2.5 temporaryImageWithCommandBuffer:imageDescriptor:()

```
+ (nonnull instancetype) temporaryImageWithCommandBuffer:
            (nonnull id< MTLCommandBuffer >) commandBuffer
            imageDescriptor:(const MPSImageDescriptor *__nonnull) imageDescriptor
```

Initialize a MPSTemporaryImage for use on a MTLCommandBuffer

Parameters

commandBuffer	The MTLCommandBuffer on which the MPSTemporaryImage will be exclusively used
imageDescriptor	A valid imageDescriptor describing the MPSImage format to create.

Returns

A valid MPSTemporaryImage. The object will be released when the command buffer is committed. The underlying texture will become invalid before this time due to the action of the readCount property.

5.154.2.6 temporaryImageWithCommandBuffer:textureDescriptor:()

```
+ (nonnull instancetype) temporaryImageWithCommandBuffer:
            (nonnull id< MTLCommandBuffer >) commandBuffer
            textureDescriptor:(const MTLTextureDescriptor *__nonnull) textureDescriptor
```

Low level interface for creating a MPSTemporaryImage using a MTLTextureDescriptor This function provides access to MTLPixelFormats not typically covered by -initForCommandBuffer:imageDescriptor: The feature channels will be inferred from the MTLPixelFormat without changing the width. The following restrictions apply:

```
MTLTextureType must be MTLTextureType2D or MTLTextureType2DArray
MTLTextureUsage must contain at least one of MTLTextureUsageShaderRead, MTLTextureUsageShaderWrite
MTLStorageMode must be MTLStorageModePrivate
depth must be 1
```

Parameters

commandBuffer	The command buffer on which the MPSTemporaryImage may be used
textureDescriptor	A texture descriptor describing the MPSTemporaryImage texture

Returns

A valid MPSTemporaryImage. The object will be released when the command buffer is committed. The underlying texture will become invalid before this time due to the action of the readCount property.

5.154.3 Property Documentation

5.154.3.1 readCount

```
- (NSUInteger) readCount [read], [write], [nonatomic], [assign]
```

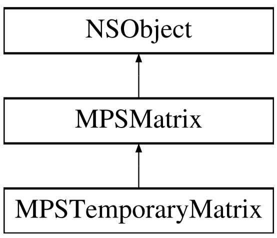
The documentation for this class was generated from the following file:

- MPSCore.framework/Headers/MPSImage.h

5.155 MPSTemporaryMatrix Class Reference

```
#import <MPSMatrixTypes.h>
```

Inheritance diagram for MPSTemporaryMatrix:



Instance Methods

- (nonnull instancetype) - initWithBuffer:descriptor:

Class Methods

- (nonnull instancetype) + temporaryMatrixWithCommandBuffer:matrixDescriptor:
- (void) + prefetchStorageWithCommandBuffer:matrixDescriptorList:

Properties

- NSUInteger readCount

5.155.1 Method Documentation

5.155.1.1 initWithBuffer:descriptor:()

```
- (nonnull instancetype) initWithBuffer:
            (nonnull id< MTLBuffer >) buffer
            descriptor:(nonnull MPSMatrixDescriptor *) descriptor
```

*** unavailable

Reimplemented from MPSMatrix.

5.155.1.2 prefetchStorageWithCommandBuffer:matrixDescriptorList:()

```
+ (void) prefetchStorageWithCommandBuffer:
            (nonnull id< MTLCommandBuffer >) commandBuffer
            matrixDescriptorList:(NSArray< MPSMatrixDescriptor * > *__nonnull) descriptor↩
List
```

Help MPS decide which allocations to make ahead of time The buffer cache that underlies the MPSTemporaryMatrix can automatically allocate new storage as needed as you create new temporary matrices. However, sometimes a more global view of what you plan to make is useful for maximizing memory reuse to get the most efficient operation. This class method hints to the cache what the list of matrices will be.

It is never necessary to call this method. It is purely a performance and memory optimization.

Parameters

commandBuffer	The command buffer on which the MPSTemporaryMatrix will be used
descriptorList	A NSArray of MPSMatrixDescriptor, indicating matrices that will be created

5.155.1.3 temporaryMatrixWithCommandBuffer:matrixDescriptor:()

```
+ (nonnull instancetype) temporaryMatrixWithCommandBuffer:
            (nonnull id< MTLCommandBuffer >) commandBuffer
            matrixDescriptor:(nonnull MPSMatrixDescriptor *) matrixDescriptor
```

Initialize a MPSTemporaryMatrix for use on a MTLCommandBuffer

Parameters

commandBuffer	The MTLCommandBuffer on which the MPSTemporaryMatrix will be exclusively used
matrixDescriptor	A valid MPSMatrixDescriptor describing the MPSMatrix format to create

Returns

A valid MPSTemporaryMatrix. The object is not managed by a NSAutoreleasePool. The object will be released when the command buffer is committed. The underlying buffer will become invalid before this time due to the action of the readCount property. Please read and understand the use of the readCount property before using this object.

5.155.2 Property Documentation

5.155.2.1 readCount

```
- (NSUInteger) readCount  [read], [write], [nonatomic], [assign]
```

The number of times a temporary matrix may be read by a MPSMatrix... kernel before its contents become undefined.

MPSTemporaryMatrices must release their underlying buffers for reuse immediately after last use. So as to facilitate *prompt* convenient memory recycling, each time a MPSTemporaryMatrix is read by a MPSMatrix... -encode... method, its readCount is automatically decremented. When the readCount reaches 0, the underlying buffer is automatically made available for reuse to MPS for its own needs and for other MPSTemporaryMatrices prior to return from the -encode.. function. The contents of the buffer become undefined at this time.

By default, the readCount is initialized to 1, indicating a matrix that may be overwritten any number of times, but read only once.

You may change the readCount as desired to allow MPSMatrixKernels to read the MPSTemporaryMatrix additional times. However, it is an error to change the readCount once it is zero. It is an error to read or write to a MPS←TemporaryMatrix with a zero readCount. You may set the readCount to 0 yourself to cause the underlying buffer to be returned to MPS. Writing to a MPSTemporaryMatrix does not adjust the readCount.

The Metal API Validation layer will assert if a MPSTemporaryMatrix is deallocated with non-zero readCount to help identify cases when resources are not returned promptly.

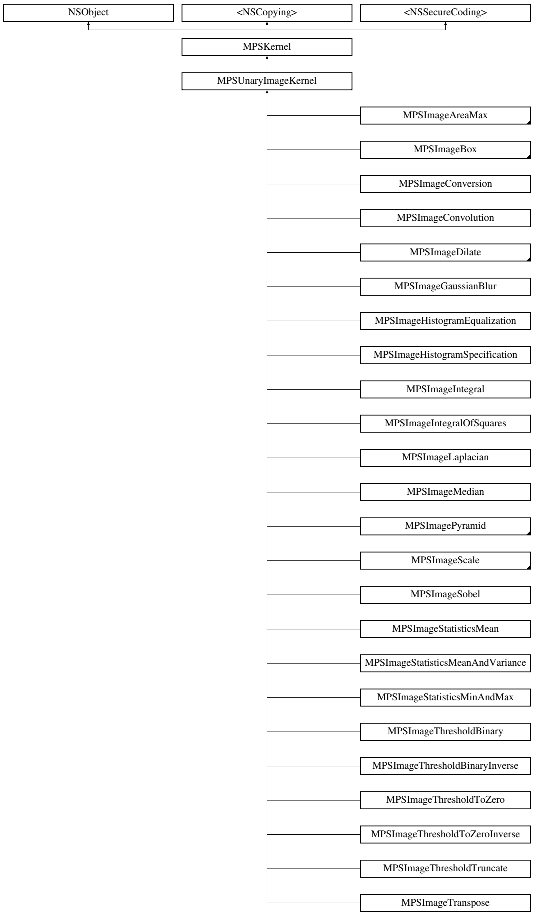
The documentation for this class was generated from the following file:

- MPSMatrixTypes.h

5.156 MPSUnaryImageKernel Class Reference

```
#import <MPSImageKernel.h>
```

Inheritance diagram for MPSUnaryImageKernel:



Instance Methods

- (nonnull instancetype) - initWithDevice:
- (nullable instancetype) - initWithCoder:device:
- (BOOL) - encodeToCommandBuffer:inPlaceTexture:fallbackCopyAllocator:
- (void) - encodeToCommandBuffer:sourceTexture:destinationTexture:
- (void) - encodeToCommandBuffer:sourceImage:destinationImage:
- (MPSRegion) - sourceRegionForDestinationSize:

Properties

- MPSOffset offset
- MTLRegion clipRect
- MPSImageEdgeMode edgeMode

Additional Inherited Members

5.156.1 Detailed Description

This depends on Metal.framework A MPSUnaryImageKernel consumes one MTLTexture and produces one MTL↩ Texture.

5.156.2 Method Documentation

5.156.2.1 encodeToCommandBuffer:inPlaceTexture:fallbackCopyAllocator:()

```
- (BOOL) encodeToCommandBuffer:
            (nonnull id< MTLCommandBuffer >) commandBuffer
            inPlaceTexture:(__nonnull id< MTLTexture > __strong *__nonnull) texture
            fallbackCopyAllocator:(nullable MPSCopyAllocator) copyAllocator
```

This method attempts to apply the MPSKernel in place on a texture.

```
In-place operation means that the same texture is used both to hold the input
image and the results. Operating in-place can be an excellent way to reduce
resource utilization, and save time and energy. While simple Metal kernels can
not operate in place because textures can not be readable and writable at the
same time, some MPSKernels can operate in place because they use
multi-pass algorithms. Whether a MPSKernel can operate in-place can
depend on current hardware, operating system revision and the parameters
and properties passed to it. You should never assume that a MPSKernel will
continue to work in place, even if you have observed it doing so before.
```

If the operation succeeds in-place, YES is returned. If the in-place operation fails and no copyAllocator is provided, then NO is returned. Without a fallback MPSCopyAllocator, in neither case is the pointer held at ∗texture modified.

Failure during in-place operation is very common and will occur inconsistently across different hardware platforms and OS releases. Without a fallback MPSCopyAllocator, operating in place may require significant error handling code to accompany each call to -encodeToCommandBuffer:..., complicating your code.

You may find it simplifies your code to provide a fallback MPSCopyAllocator so that the operation can proceed reliably even when it can not proceed in-place. When an in-place filter fails, the MPSCopyAllocator (if any) will be invoked to create a new texture in which to write the results, allowing the filter to proceed reliably out-of-place. The original texture will be released, replaced with a pointer to the new texture and YES will be returned. If the allocator returns an invalid texture, it is released, ∗texture remains unmodified and NO is returned. Please see the MPSCopyAllocator definition for a sample allocator implementation.

Sample usage with a copy allocator:

```
id <MTLTexture> inPlaceTex = ...;
MPSImageSobel *sobelFiler = [[MPSImageSobel alloc] initWithDevice: my_device];

// With a fallback MPSCopyAllocator, failure should only occur in exceptional
// conditions such as MTLTexture allocation failure or programmer error.
// That is, the operation is roughly as robust as the MPSCopyAllocator.
// Depending on the quality of that, we might decide we are justified here
// in not checking the return value.
[sobelFilter encodeToCommandBuffer: my_command_buffer
                    inPlaceTexture: &inPlaceTex  // may be replaced!
             fallbackCopyAllocator: myAllocator];

// If myAllocator was not called:
//
//      inPlaceTex holds the original texture with the result pixels in it
//
// else,
//
//      1) myAllocator creates a new texture.
//      2) The new texture pixel data is overwritten by MPSUnaryImageKernel.
//      3) The old texture passed in *inPlaceTex is released once.
//      4) *inPlaceTex = the new texture
//
// In either case, the caller should now hold one reference to the texture now held in
// inPlaceTex, whether it was replaced or not. Most of the time that means that nothing
// further needs to be done here, and you can proceed to the next image encoding operation.
// However, if other agents held references to the original texture, they still hold them
// and may need to be alerted that the texture has been replaced so that they can retain
// the new texture and release the old one.

[sobelFilter release];  // if not ARC, clean up the MPSImageSobel object
```

Note: Image filters that look at neighboring pixel values may actually consume more memory when operating in place than out of place. Many such operations are tiled internally to save intermediate texture storage, but can not tile when operating in place. The memory savings for tiling is however very short term, typically the lifetime of the MTLCommandBuffer.

Attempt to apply a MPSKernel to a texture in place.

Parameters

commandBuffer	A valid MTLCommandBuffer to receive the encoded filter
texture	A pointer to a valid MTLTexture containing source image. On success, the image contents and possibly texture itself will be replaced with the result image.
copyAllocator	An optional block to allocate a new texture to hold the results, in case in-place operation is not possible. The allocator may use a different MTLPixelFormat or size than the original texture. You may enqueue operations on the provided MTLCommandBuffer using the provided MTLComputeCommandEncoder to initialize the texture contents.

Returns

On success, YES is returned. The texture may have been replaced with a new texture if a copyAllocator was provided. On failure, NO is returned. The texture is unmodified.

5.156.2.2 encodeToCommandBuffer:sourceImage:destinationImage:()

```
- (void) encodeToCommandBuffer:
            (nonnull id< MTLCommandBuffer >) commandBuffer
            sourceImage:(nonnull MPSImage *) sourceImage
            destinationImage:(nonnull MPSImage *) destinationImage
```

Encode a MPSKernel into a command Buffer. The operation shall proceed out-of-place.

Parameters

commandBuffer	A valid MTLCommandBuffer to receive the encoded filter
sourceImage	A valid MPSImage containing the source image.
destinationImage	A valid MPSImage to be overwritten by result image. DestinationImage may not alias sourceImage.

5.156.2.3 encodeToCommandBuffer:sourceTexture:destinationTexture:()

```
- (void) encodeToCommandBuffer:
            (nonnull id< MTLCommandBuffer >) commandBuffer
            sourceTexture:(nonnull id< MTLTexture >) sourceTexture
            destinationTexture:(nonnull id< MTLTexture >) destinationTexture
```

Encode a MPSKernel into a command Buffer. The operation shall proceed out-of-place.

Parameters

commandBuffer	A valid MTLCommandBuffer to receive the encoded filter
sourceTexture	A valid MTLTexture containing the source image.
destinationTexture	A valid MTLTexture to be overwritten by result image. DestinationTexture may not alias sourceTexture.

5.156.2.4 initWithCoder:device:()

```
- (nullable instancetype) initWithCoder:
            (NSCoder *__nonnull) aDecoder
            device:(nonnull id< MTLDevice >) device
```

NSSecureCoding compatability While the standard NSSecureCoding/NSCoding method -initWithCoder: should work, since the file can't know which device your data is allocated on, we have to guess and may guess incorrectly. To avoid that problem, use initWithCoder:device instead.

Parameters

aDecoder	The NSCoder subclass with your serialized MPSKernel
device	The MTLDevice on which to make the MPSKernel

Returns

A new MPSKernel object, or nil if failure.

Reimplemented from MPSKernel.

Reimplemented in MPSImagePyramid, MPSImageSobel, MPSImageHistogramSpecification, MPSImageThresholdToZeroInverse, MPSImageThresholdToZero, MPSImageHistogramEqualization, MPSImageBox, MPSImageGaussianBlur, MPSImageStatisticsMean, MPSImageThresholdBinary, MPSImageThresholdTruncate, MPSImageDilate, MPSImageScale, MPSImageLanczosScale, MPSImageBilinearScale, MPSImageStatisticsMeanAndVariance, MPSImageConvolution, MPSImageThresholdBinaryInverse, MPSImageStatisticsMinAndMax, MPSImageMedian, and MPSImageAreaMax.

5.156.2.5 initWithDevice:()

```
- (nonnull instancetype) initWithDevice:
            (nonnull id< MTLDevice >) device
```

Standard init with default properties per filter type

Parameters

device	The device that the filter will be used on. May not be NULL.

Returns

a pointer to the newly initialized object. This will fail, returning nil if the device is not supported. Devices must be MTLFeatureSet_iOS_GPUFamily2_v1 or later.

Reimplemented from MPSKernel.

Reimplemented in MPSImagePyramid, MPSImageSobel, MPSImageThresholdToZeroInverse, MPSImageThresholdToZero, MPSImageBox, MPSImageGaussianBlur, MPSImageStatisticsMean, MPSImageThresholdBinary, MPSImageThresholdTruncate, MPSImageDilate, MPSImageScale, MPSImageLanczosScale, MPSImageBilinearScale, MPSImageStatisticsMeanAndVariance, MPSImageThresholdBinaryInverse, MPSImageStatisticsMinAndMax, MPSImageMedian, and MPSImageAreaMax.

5.156.2.6 sourceRegionForDestinationSize:()

```
- (MPSRegion) sourceRegionForDestinationSize:
            (MTLSize) destinationSize
```

sourceRegionForDestinationSize: is used to determine which region of the sourceTexture will be read by encodeToCommandBuffer:sourceTexture:destinationTexture (and similar) when the filter runs. This information may be needed if the source image is broken into multiple textures. The size of the full (untiled) destination image is provided. The region of the full (untiled) source image that will be read is returned. You can then piece together an appropriate texture containing that information for use in your tiled context.

The function will consult the MPSUnaryImageKernel offset and clipRect parameters, to determine the full region read by the function. Other parameters such as sourceClipRect, kernelHeight and kernelWidth will be consulted as necessary. All properties should be set to intended values prior to calling sourceRegionForDestinationSize:.

```
Caution: This function operates using global image coordinates, but
-encodeToCommandBuffer:... uses coordinates local to the source and
destination image textures. Consequently, the offset and clipRect
attached to this object will need to be updated using a global to
local coordinate transform before -encodeToCommandBuffer:... is
called.
```

Determine the region of the source texture that will be read for a encode operation

Parameters

destinationSize	The size of the full virtual destination image.

Returns

The area in the virtual source image that will be read.

5.156.3 Property Documentation

5.156.3.1 clipRect

```
- clipRect  [read], [write], [nonatomic], [assign]
```

An optional clip rectangle to use when writing data. Only the pixels in the rectangle will be overwritten. A MTL↩ Region that indicates which part of the destination to overwrite. If the clipRect does not lie completely within the destination image, the intersection between clip rectangle and destination bounds is used. Default: MPSRectNoClip (MPSKernel::MPSRectNoClip) indicating the entire image.

See Also: MetalPerformanceShaders.h subsubsection_clipRect

5.156.3.2 edgeMode

```
- edgeMode  [read], [write], [nonatomic], [assign]
```

The MPSImageEdgeMode to use when texture reads stray off the edge of an image Most MPSKernel objects can read off the edge of the source image. This can happen because of a negative offset property, because the offset + clipRect.size is larger than the source image or because the filter looks at neighboring pixels, such as a Convolution or morphology filter. Default: usually MPSImageEdgeModeZero. (Some MPSKernel types default to MPSImageEdgeModeClamp, because MPSImageEdgeModeZero is either not supported or would produce unexpected results.)

See Also: MetalPerformanceShaders.h subsubsection_edgemode

5.156.3.3 offset

```
- offset  [read], [write], [nonatomic], [assign]
```

The position of the destination clip rectangle origin relative to the source buffer. The offset is defined to be the position of clipRect.origin in source coordinates. Default: {0,0,0}, indicating that the top left corners of the clipRect and source image align.

See Also: MetalPerformanceShaders.h subsubsection_mpsoffset

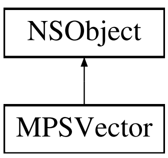
The documentation for this class was generated from the following file:

- MPSImageKernel.h

5.157 MPSVector Class Reference

```
#import <MPSMatrixTypes.h>
```

Inheritance diagram for MPSVector:



Instance Methods

- (nonnull instancetype) - initWithBuffer:descriptor:
- (nonnull instancetype) - init

Properties

- id< MTLDevice > device
- NSUInteger length
- NSUInteger vectors
- MPSDataType dataType
- NSUInteger vectorBytes
- id< MTLBuffer > data

5.157.1 Detailed Description

This depends on Metal.framework

A MPSVector object describes a 1-dimensional array of data and provides storage for its values. Some MPS↩ MatrixKernel objects operate on MPSVector objects for convenience.

5.157.2 Method Documentation

5.157.2.1 init()

```
- (nonnull instancetype) init
```

5.157.2.2 initWithBuffer:descriptor:()

```
- (nonnull instancetype) initWithBuffer:
            (nonnull id< MTLBuffer >) buffer
            descriptor:(nonnull MPSVectorDescriptor *) descriptor
```

Initialize a MPSVector object with a MTLBuffer.

Parameters

buffer	The MTLBuffer object which contains the data to use for the MPSVector. May not be NULL.
descriptor	The MPSVectorDescriptor. May not be NULL.

Returns

A valid MPSVector object or nil, if failure.

This function returns a MPSVector object which uses the supplied MTLBuffer. The length, number of vectors, and stride between vectors are specified by the MPSVectorDescriptor object.

The provided MTLBuffer must have enough storage to hold

```
(descriptor.vectors-1) * descriptor.vectorBytes +
 descriptor.length * (element size) bytes.
```

5.157.3 Property Documentation

5.157.3.1 data

```
- data  [read], [nonatomic], [assign]
```

An MTLBuffer to store the data.

5.157.3.2 dataType

```
- dataType  [read], [nonatomic], [assign]
```

The type of the MPSVector data.

5.157.3.3 device

```
- device  [read], [nonatomic], [retain]
```

The device on which the MPSVector will be used.

5.157.3.4 length

```
- length  [read], [nonatomic], [assign]
```

The number of elements in the vector.

5.157.3.5 vectorBytes

```
- vectorBytes  [read], [nonatomic], [assign]
```

The stride, in bytes, between corresponding elements of consecutive vectors.

5.157.3.6 vectors

```
- vectors  [read], [nonatomic], [assign]
```

The number of vectors in the MPSVector.

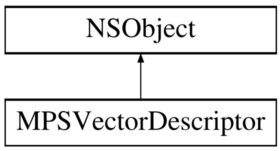
The documentation for this class was generated from the following file:

- MPSMatrixTypes.h

5.158 MPSVectorDescriptor Class Reference

```
#import <MPSMatrixTypes.h>
```

Inheritance diagram for MPSVectorDescriptor:



Class Methods

- (__nonnull instancetype) + vectorDescriptorWithLength:dataType:
- (__nonnull instancetype) + vectorDescriptorWithLength:vectors:vectorBytes:dataType:
- (size_t) + vectorBytesForLength:dataType:

Properties

- NSUInteger length
- NSUInteger vectors
- MPSDataType dataType
- NSUInteger vectorBytes

5.158.1 Detailed Description

This depends on Metal.framework

A MPSVectorDescriptor describes the length and data type of a an array of 1-dimensional vectors. All vectors are stored as contiguous arrays of data.

5.158.2 Method Documentation

5.158.2.1 vectorBytesForLength:dataType:()

```
+ (size_t) vectorBytesForLength:
            (NSUInteger) length
            dataType:(MPSDataType) dataType
```

Return the recommended stride, in bytes, to be used for an array of vectors of a given length.

Parameters

length	The number of elements in a single vector.
dataType	The type of vector data values.

To achieve best performance the optimal stride between vectors within an array of vectors is not necessarily equivalent to the number of elements per vector. This method returns the stride, in bytes, which gives best performance for a given vector length. Using this stride to construct your array is recommended, but not required (provided that the stride used is still large enough to allocate a full vector of data).

5.158.2.2 vectorDescriptorWithLength:dataType:()

```
+ (__nonnull instancetype) vectorDescriptorWithLength:
            (NSUInteger) length
            dataType:(MPSDataType) dataType
```

Create a MPSVectorDescriptor with the specified length and data type.

Parameters

length	The number of elements in a single vector.
dataType	The type of the data to be stored in the vector.

Use this function for creating a descriptor of a MPSVector object containing a single vector.

5.158.2.3 vectorDescriptorWithLength:vectors:vectorBytes:dataType:()

```
+ (__nonnull instancetype) vectorDescriptorWithLength:
            (NSUInteger) length
            vectors:(NSUInteger) vectors
            vectorBytes:(NSUInteger) vectorBytes
            dataType:(MPSDataType) dataType
```

Create a MPSVectorDescriptor with the specified length and data type.

Parameters

length	The number of elements in a single vector.
vectors	The number of vectors in the MPSVector object.
vectorBytes	The number of bytes between starting elements of consecutive vectors.
dataType	The type of the data to be stored in the vector.

For performance considerations the optimal stride between vectors may not necessarily be equal to the vector length. The MPSVectorDescriptor class provides a method which may be used to determine this value, see the vectorBytesForLength API.

5.158.3 Property Documentation

5.158.3.1 dataType

```
- dataType  [read], [write], [nonatomic], [assign]
```

The type of the data which makes up the values of the vector.

5.158.3.2 length

```
- length  [read], [write], [nonatomic], [assign]
```

The number of elements in the vector.

5.158.3.3 vectorBytes

```
- vectorBytes  [read], [nonatomic], [assign]
```

The stride, in bytes, between corresponding elements of consecutive vectors. Must be a multiple of the element size

5.158.3.4 vectors

```
- vectors  [read], [nonatomic], [assign]
```

The number of vectors.

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

- MPSMatrixTypes.h

Chapter 6

File Documentation

6.1 MetalPerformanceShaders.h File Reference

```
#import <MPSCore/MPSCore.h>
#import <MPSImage/MPSImage.h>
#import <MPSMatrix/MPSMatrix.h>
#import <MPSNeuralNetwork/MPSNeuralNetwork.h>
```

Functions

- BOOL MPSSupportsMTLDevice (__nullable id< MTLDevice > device) ""

6.1.1 Function Documentation

6.1.1.1 MPSSupportsMTLDevice()

```
BOOL MPSSupportsMTLDevice (
            __nullable id< MTLDevice > device )
```

MetalPerformanceShaders.h MetalPerformanceShaders

Copyright

MPSSupportsMTLDevice Determine whether a MetalPerformanceShaders.framework supports a MTLDevice. Use this function to determine whether a MTLDevice can be used with interfaces in MetalPerformanceShaders.↩ framework.

Parameters

device	A valid MTLDevice

Returns

YES The device is supported. NO The device is not supported

6.2 MPSCNNConvolution.h File Reference

```
#include <MPSNeuralNetwork/MPSCNNKernel.h>
#include <MPSNeuralNetwork/MPSCNNNormalization.h>
#include <MPSNeuralNetwork/MPSCNNNeuronType.h>
#include <MPSCore/MPSState.h>
#include <simd/simd.h>
```

Classes

- class MPSCNNNeuron
- class MPSCNNNeuronLinear
- class MPSCNNNeuronReLU
- class MPSCNNNeuronSigmoid
- class MPSCNNNeuronHardSigmoid
- class MPSCNNNeuronTanH
- class MPSCNNNeuronAbsolute
- class MPSCNNNeuronSoftPlus
- class MPSCNNNeuronSoftSign
- class MPSCNNNeuronELU
- class MPSCNNConvolutionDescriptor
- class MPSCNNSubPixelConvolutionDescriptor
- protocol <MPSCNNConvolutionDataSource >
- class MPSCNNConvolutionState
- class MPSCNNConvolution
- class MPSCNNFullyConnected
- class MPSCNNConvolutionTranspose
- class MPSCNNBinaryConvolution
- class MPSCNNBinaryFullyConnected

6.3 MPSCNNKernel.h File Reference

```
#include <MPSCore/MPSKernel.h>
#include <MPSCore/MPSImage.h>
#include <MPSNeuralNetwork/MPSNeuralNetworkTypes.h>
```

Classes

- class MPSCNNKernel
- class MPSCNNBinaryKernel

6.4 MPSCNNNeuronType.h File Reference

Macros

- #define MPS_SWIFT_NAME(_a)
- #define MPS_ENUM_AVAILABLE_STARTING(...)

Typedefs

- typedef enum MPSCNNNeuronType MPSCNNNeuronType

Enumerations

- enum MPSCNNNeuronType {
MPSCNNNeuronTypeNone, MPSCNNNeuronTypeReLU, MPSCNNNeuronTypeLinear, MPSCNNNeuron←TypeSigmoid,
MPSCNNNeuronTypeHardSigmoid, MPSCNNNeuronTypeTanH, MPSCNNNeuronTypeAbsolute, MPSCN←NNeuronTypeSoftPlus,
MPSCNNNeuronTypeSoftSign, MPSCNNNeuronTypeELU, MPSCNNNeuronTypeCount }

6.4.1 Macro Definition Documentation

6.4.1.1 MPS_ENUM_AVAILABLE_STARTING

```
#define MPS_ENUM_AVAILABLE_STARTING(
              ... )
```

6.4.1.2 MPS_SWIFT_NAME

```
#define MPS_SWIFT_NAME(
              _a )
```

6.4.2 Typedef Documentation

6.4.2.1 MPSCNNNeuronType

```
typedef enum MPSCNNNeuronType
MPSCNNNeuronType
```

6.4.3 Enumeration Type Documentation

6.4.3.1 MPSCNNNeuronType

```
enum MPSCNNNeuronType
```

Enumerator

MPSCNNNeuronTypeNone	
MPSCNNNeuronTypeReLU	
MPSCNNNeuronTypeLinear	
MPSCNNNeuronTypeSigmoid	
MPSCNNNeuronTypeHardSigmoid	
MPSCNNNeuronTypeTanH	
MPSCNNNeuronTypeAbsolute	
MPSCNNNeuronTypeSoftPlus	
MPSCNNNeuronTypeSoftSign	
MPSCNNNeuronTypeELU	
MPSCNNNeuronTypeCount	

6.5 MPSCNNNormalization.h File Reference

```
#include <MPSNeuralNetwork/MPSCNNKernel.h>
```

Classes

- class MPSCNNSpatialNormalization
- class MPSCNNLocalContrastNormalization
- class MPSCNNCrossChannelNormalization

6.6 MPSCNNPooling.h File Reference

```
#import <MPSNeuralNetwork/MPSCNNKernel.h>
#import <MPSCore/MPSCore.h>
```

Classes

- class MPSCNNPooling
- class MPSCNNPoolingMax
- class MPSCNNPoolingAverage
- class MPSCNNPoolingL2Norm
- class MPSCNNDilatedPoolingMax

6.7 MPSCNNSoftMax.h File Reference

```
#include <MPSNeuralNetwork/MPSCNNKernel.h>
```

Classes

- class MPSCNNSoftMax
- class MPSCNNLogSoftMax

6.8 MPSCNNUpsampling.h File Reference

```
#import <MPSNeuralNetwork/MPSCNNKernel.h>
```

Classes

- class MPSCNNUpsampling
- class MPSCNNUpsamplingNearest
- class MPSCNNUpsamplingBilinear

6.9 MPSCore.h File Reference

```
#import <MPSCore/MPSCoreTypes.h>
#import <MPSCore/MPSImage.h>
#import <MPSCore/MPSKernel.h>
#import <MPSCore/MPSState.h>
```

6.10 MPSCoreTypes.h File Reference

```
#import <Foundation/NSObject.h>
#import <Metal/Metal.h>
```

Classes

- struct MPSOffset
- struct MPSOrigin
- struct MPSSize
- struct MPSRegion
- protocol <MPSDeviceProvider>

Macros

- #define __has_attribute(a) 0
- #define __has_feature(f) 0
- #define __has_extension(e) 0
- #define MPS_HIDE_AVAILABILITY 1
- #define MPS_ENUM_AVAILABLE_STARTING(...)
- #define MPS_ENUM_AVAILABLE_STARTING_BUT_DEPRECATED(...)
- #define MPS_CLASS_AVAILABLE_STARTING(...)
- #define MPS_AVAILABLE_STARTING(...)
- #define MPS_AVAILABLE_STARTING_BUT_DEPRECATED(...)
- #define MPS_SWIFT_NAME(...)

Typedefs

- typedef enum MPSImageEdgeMode MPSImageEdgeMode
- typedef enum MPSImageFeatureChannelFormat MPSImageFeatureChannelFormat
- typedef enum MPSDataType MPSDataType
- typedef struct MPSOrigin MPSOrigin
- typedef struct MPSSize MPSSize
- typedef struct MPSRegion MPSRegion

Enumerations

- enum MPSKernelOptions {
  MPSKernelOptionsNone, MPSKernelOptionsSkipAPIValidation, MPSKernelOptionsAllowReducedPrecision,
  MPSKernelOptionsDisableInternalTiling,
  MPSKernelOptionsInsertDebugGroups, MPSKernelOptionsVerbose }
- enum MPSImageEdgeMode { MPSImageEdgeModeZero, MPSImageEdgeModeClamp }
- enum MPSImageFeatureChannelFormat {
  MPSImageFeatureChannelFormatNone, MPSImageFeatureChannelFormatUnorm8, MPSImageFeature↩
  ChannelFormatUnorm16, MPSImageFeatureChannelFormatFloat16,
  MPSImageFeatureChannelFormatFloat32 }
- enum MPSDataType {
  MPSDataTypeInvalid, MPSDataTypeFloatBit, MPSDataTypeFloat32, MPSDataTypeFloat16,
  MPSDataTypeSignedBit, DEPRECATED_ATTRIBUTE = MPSDataTypeSignedBit, MPSDataTypeInt8, MP↩
  SDataTypeInt16,
  MPSDataTypeUInt8, MPSDataTypeUInt16, MPSDataTypeUInt32, MPSDataTypeNormalizedBit,
  MPSDataTypeUnorm1, MPSDataTypeUnorm8 }

6.10.1 Macro Definition Documentation

6.10.1.1 __has_attribute

```
#define __has_attribute(
              a ) 0
```

MPSTypes.h MPSCore

Copyright

 Types common to MetalPerformanceShaders.framework

6.10.1.2 __has_extension

```
#define __has_extension(
              e ) 0
```

6.10.1.3 __has_feature

```
#define __has_feature(
              f ) 0
```

6.10.1.4 MPS_AVAILABLE_STARTING

```
#define MPS_AVAILABLE_STARTING(
              ... )
```

6.10.1.5 MPS_AVAILABLE_STARTING_BUT_DEPRECATED

```
#define MPS_AVAILABLE_STARTING_BUT_DEPRECATED(
              ... )
```

6.10.1.6 MPS_CLASS_AVAILABLE_STARTING

```
#define MPS_CLASS_AVAILABLE_STARTING(
              ... )
```

6.10.1.7 MPS_ENUM_AVAILABLE_STARTING

```
#define MPS_ENUM_AVAILABLE_STARTING(
              ... )
```

6.10.1.8 MPS_ENUM_AVAILABLE_STARTING_BUT_DEPRECATED

```
#define MPS_ENUM_AVAILABLE_STARTING_BUT_DEPRECATED(
              ... )
```

6.10.1.9 MPS_HIDE_AVAILABILITY

```
#define MPS_HIDE_AVAILABILITY 1
```

6.10.1.10 MPS_SWIFT_NAME

```
#define MPS_SWIFT_NAME(
              ... )
```

6.10.2 Typedef Documentation

6.10.2.1 MPSDataType

```
typedef enum MPSDataType
```

MPSDataType

6.10.2.2 MPSImageEdgeMode

```
typedef enum MPSImageEdgeMode
```

MPSImageEdgeMode

6.10.2.3 MPSImageFeatureChannelFormat

```
typedef enum MPSImageFeatureChannelFormat
```

MPSImageFeatureChannelFormat

6.10.2.4 MPSOrigin

```
typedef struct MPSOrigin MPSOrigin
```

6.10.2.5 MPSRegion

```
typedef struct MPSRegion MPSRegion
```

6.10.2.6 MPSSize

```
typedef struct MPSSize MPSSize
```

6.10.3 Enumeration Type Documentation

6.10.3.1 MPSDataType

```
enum MPSDataType
```

A value to specify a type of data.

MPSDataTypeFloatBit A common bit for all floating point data types. Zero for integer types MPSDataType↩ NormalizedBit If set, the value of the shall be interpreted as value / UNORM_TYPE_MAX Normalized values have range [0, 1.0] if unsigned and [-1,1] if signed. SNORM_TYPE_MIN is interpreted as SNORM_TYPE_MIN+1 per standard Metal rules.

MSPDataTypeFloat32 32-bit floating point (single-precision). MSPDataTypeFloat16 16-bit floating point (half-precision). (IEEE-754-2008 float16 exchange format) MPSDataTypeInt8 Signed 8-bit integer. MPSDataTypeInt16 Signed 16-bit integer. MPSDataTypeUInt8 Unsigned 8-bit integer. Not normalized MPSDataTypeUInt16 Unsigned 16-bit integer. Not normalized MPSDataTypeUInt32 Unsigned 32-bit integer. Not normalized MPSDataTypeUnorm1 Unsigned 1-bit normalized value. MPSDataTypeUnorm8 Unsigned 8-bit normalized value.

Enumerator

MPSDataTypeInvalid	
MPSDataTypeFloatBit	
MPSDataTypeFloat32	
MPSDataTypeFloat16	
MPSDataTypeSignedBit	
DEPRECATED_ATTRIBUTE	
MPSDataTypeInt8	
MPSDataTypeInt16	
MPSDataTypeUInt8	
MPSDataTypeUInt16	
MPSDataTypeUInt32	
MPSDataTypeNormalizedBit	
MPSDataTypeUnorm1	
MPSDataTypeUnorm8	

6.10.3.2 MPSImageEdgeMode

```
enum MPSImageEdgeMode
```

Options used to control edge behaviour of filter when filter reads beyond boundary of src image

Enumerator

MPSImageEdgeModeZero	Out of bound pixels are (0,0,0,1) for image with pixel format without alpha channel and (0,0,0,0) for image with pixel format that has an alpha channel
MPSImageEdgeModeClamp	Out of bound pixels are clamped to nearest edge pixel

6.10.3.3 MPSImageFeatureChannelFormat

```
enum MPSImageFeatureChannelFormat
```

Encodes the representation of a single channel within a MPSImage. A MPSImage pixel may have many channels in it, sometimes many more than 4, the limit of what MTLPixelFormats encode. The storage format for a single channel within a pixel can be given by the MPSImageFeatureChannelFormat. The number of channels is given by the featureChannels parameter of appropriate MPSImage APIs. The size of the pixel is size of the channel format multiplied by the number of feature channels. No padding is allowed, except to round out to a full byte.

Enumerator

MPSImageFeatureChannelFormatNone	No format. This can mean according to context invalid format or any format. In the latter case, it is an invitation to MPS to pick a format.
MPSImageFeatureChannelFormatUnorm8	uint8_t with value [0,255] encoding [0,1.0]
MPSImageFeatureChannelFormatUnorm16	uint16_t with value [0,65535] encoding [0,1.0]
MPSImageFeatureChannelFormatFloat16	IEEE-754 16-bit floating-point value. "half precision" Representable normal range is +-[2∗∗-14, 65504], 0, Infinity, NaN. 11 bits of precision + exponent.
MPSImageFeatureChannelFormatFloat32	IEEE-754 32-bit floating-point value. "single precision" (standard float type in C) 24 bits of precision + exponent

6.10.3.4 MPSKernelOptions

```
enum MPSKernelOptions
```

Options used when creating MPSKernel objects

Enumerator

MPSKernelOptionsNone	Use default options
MPSKernelOptionsSkipAPIValidation	Most MPS functions will sanity check their arguments. This has a small but non-zero CPU cost. Setting the MPSKernelOptionsSkipAPIValidation will skip these checks. MPSKernelOptionsSkipAPIValidation does not skip checks for memory allocation failure. Caution: turning on MPSKernelOptionsSkipAPIValidation can result in undefined behavior if the requested operation can not be completed for some reason. Most error states will be passed through to Metal which may do nothing or abort the program if Metal API validation is turned on.

Enumerator

MPSKernelOptionsAllowReducedPrecision	When possible, MPSKernels use a higher precision data representation internally than the destination storage format to avoid excessive accumulation of computational rounding error in the result. MPSKernelOptionsAllowReducedPrecision advises the MPSKernel that the destination storage format already has too much precision for what is ultimately required downstream, and the MPSKernel may use reduced precision internally when it feels that a less precise result would yield better performance. The expected performance win is often small, perhaps 0-20%. When enabled, the precision of the result may vary by hardware and operating system.
MPSKernelOptionsDisableInternalTiling	Some MPSKernels may automatically split up the work internally into multiple tiles. This improves performance on larger textures and reduces the amount of memory needed by MPS for temporary storage. However, if you are using your own tiling scheme to achieve similar results, your tile sizes and MPS's choice of tile sizes may interfere with one another causing MPS to subdivide your tiles for its own use inefficiently. Pass MPSKernelOptionsDisableInternalTiling to force MPS to process your data tile as a single chunk.
MPSKernelOptionsInsertDebugGroups	Enabling this bit will cause various -encode... methods to call MTLCommandEncoder push/popDebugGroup. The debug string will be drawn from MPSKernel.label, if any or the name of the class otherwise.
MPSKernelOptionsVerbose	Some parts of MPS can provide debug commentary and tuning advice when run. Setting this bit to 1 will cause the commentary to be emitted to stderr. Otherwise, the code is silent. This is especially useful for debugging MPSNNGraph. This option is on by default when the MPS_LOG_INFO environment variable is defined. For even more detailed output on a MPS object, you can use the po command in llvm with MPS objects: `llvm> po <MPS object pointer>`

6.11 MPSImage.h File Reference

```
#include <MPSCore/MPSCoreTypes.h>
#import <Metal/MTLBuffer.h>
```

Classes

- class MPSImageDescriptor
- protocol <MPSImageAllocator >
- struct MPSImageReadWriteParams
- class MPSImage
- class MPSTemporaryImage

Enumerations

- enum MPSPurgeableState
- enum MPSDataLayout

Functions

- enum MPSPurgeableState NS_ENUM_AVAILABLE (10_11, 8_0) MPSPurgeableState
- enum MPSDataLayout NS_ENUM_AVAILABLE (10_13, 11_0) MPSDataLayout

Variables

- typedef NS_ENUM_AVAILABLE
- MPSPurgeableStateAllocationDeferred
- MPSPurgeableStateKeepCurrent
- MPSPurgeableStateNonVolatile
- MPSPurgeableStateVolatile
- MPSPurgeableStateEmpty
- MPSDataLayoutHeightxWidthxFeatureChannels
- MPSDataLayoutFeatureChannelsxHeightxWidth

6.11.1 Enumeration Type Documentation

6.11.1.1 MPSDataLayout

```
enum MPSDataLayout
```

6.11.1.2 MPSPurgeableState

```
enum MPSPurgeableState
```

6.11.2 Function Documentation

6.11.2.1 NS_ENUM_AVAILABLE() [1/2]

```
enum MPSPurgeableState NS_ENUM_AVAILABLE (
            10_11 ,
            8_0 )
```

6.11.2.2 NS_ENUM_AVAILABLE() [2/2]

```
enum MPSDataLayout NS_ENUM_AVAILABLE (
            10_13 ,
            11_0  )
```

6.11.3 Variable Documentation

6.11.3.1 MPSDataLayoutFeatureChannelsxHeightxWidth

```
MPSDataLayoutFeatureChannelsxHeightxWidth
```

6.11.3.2 MPSDataLayoutHeightxWidthxFeatureChannels

```
MPSDataLayoutHeightxWidthxFeatureChannels
```

6.11.3.3 MPSPurgeableStateAllocationDeferred

```
MPSPurgeableStateAllocationDeferred
```

6.11.3.4 MPSPurgeableStateEmpty

```
MPSPurgeableStateEmpty
```

6.11.3.5 MPSPurgeableStateKeepCurrent

```
MPSPurgeableStateKeepCurrent
```

6.11.3.6 MPSPurgeableStateNonVolatile

```
MPSPurgeableStateNonVolatile
```

6.11.3.7 MPSPurgeableStateVolatile

MPSPurgeableStateVolatile

6.11.3.8 NS_ENUM_AVAILABLE

typedef NS_ENUM_AVAILABLE

6.12 MPSImage.h File Reference

```
#import <MPSCore/MPSKernel.h>
#import <MPSImage/MPSImageTypes.h>
#import <MPSImage/MPSImageConversion.h>
#import <MPSImage/MPSImageConvolution.h>
#import <MPSImage/MPSImageCopy.h>
#import <MPSImage/MPSImageKeypoint.h>
#import <MPSImage/MPSImageHistogram.h>
#import <MPSImage/MPSImageIntegral.h>
#import <MPSImage/MPSImageMath.h>
#import <MPSImage/MPSImageMedian.h>
#import <MPSImage/MPSImageMorphology.h>
#import <MPSImage/MPSImageResampling.h>
#import <MPSImage/MPSImageStatistics.h>
#import <MPSImage/MPSImageThreshold.h>
#import <MPSImage/MPSImageTranspose.h>
```

6.13 MPSImageConversion.h File Reference

```
#include <MPSImage/MPSImageKernel.h>
#include <CoreGraphics/CGColorConversionInfo.h>
```

Classes

- class MPSImageConversion

6.14 MPSImageConvolution.h File Reference

```
#include <MPSImage/MPSImageKernel.h>
```

Classes

- class MPSImageConvolution
- class MPSImageLaplacian
- class MPSImageBox
- class MPSImageTent
- class MPSImageGaussianBlur
- class MPSImageSobel
- class MPSImagePyramid
- class MPSImageGaussianPyramid

6.15 MPSImageCopy.h File Reference

```
#include <MPSCore/MPSImage.h>
#include <MPSImage/MPSImageKernel.h>
#include <MPSMatrix/MPSMatrix.h>
#include <simd/simd.h>
```

Classes

- class MPSImageCopyToMatrix

6.16 MPSImageHistogram.h File Reference

```
#include <MPSImage/MPSImageKernel.h>
#include <simd/simd.h>
```

Classes

- struct MPSImageHistogramInfo

    Specifies information to compute the histogram for channels of an image.
- class MPSImageHistogram
- class MPSImageHistogramEqualization
- class MPSImageHistogramSpecification

6.17 MPSImageIntegral.h File Reference

```
#include <MPSImage/MPSImageKernel.h>
```

Classes

- class MPSImageIntegral
- class MPSImageIntegralOfSquares

6.18 MPSImageKernel.h File Reference

```
#import <MPSCore/MPSKernel.h>
#import <MPSImage/MPSImageTypes.h>
```

Classes

- class MPSUnaryImageKernel
- class MPSBinaryImageKernel

Typedefs

- typedef id< MTLTexture > __nonnull NS_RETURNS_RETAINED(^ MPSCopyAllocator) (MPSKernel *__↩ nonnull filter, id< MTLCommandBuffer > __nonnull commandBuffer, id< MTLTexture > __nonnull source↩ Texture)

6.18.1 Typedef Documentation

6.18.1.1 MPSCopyAllocator

```
typedef id<MTLTexture> __nonnull NS_RETURNS_RETAINED(^ MPSCopyAllocator) (MPSKernel *__nonnull
filter, id< MTLCommandBuffer > __nonnull commandBuffer, id< MTLTexture > __nonnull source↩
Texture)
```

6.19 MPSImageKeypoint.h File Reference

```
#include <MPSImage/MPSImageKernel.h>
#include <simd/simd.h>
```

Classes

- struct MPSImageKeypointRangeInfo

  Specifies information to find the keypoints in an image.
- struct MPSImageKeypointData

  Specifies keypoint information.
- class MPSImageFindKeypoints

6.20 MPSImageMath.h File Reference

```
#include <MPSImage/MPSImageKernel.h>
#include <simd/simd.h>
```

Classes

- class MPSImageArithmetic
- class MPSImageAdd
- class MPSImageSubtract
- class MPSImageMultiply
- class MPSImageDivide

6.21 MPSImageMedian.h File Reference

```
#include <MPSImage/MPSImageKernel.h>
```

Classes

- class MPSImageMedian

6.22 MPSImageMorphology.h File Reference

```
#include <MPSImage/MPSImageKernel.h>
```

Classes

- class MPSImageAreaMax
- class MPSImageAreaMin
- class MPSImageDilate
- class MPSImageErode

6.23 MPSImageResampling.h File Reference

```
#include <MPSImage/MPSImageKernel.h>
```

Classes

- class MPSImageScale
- class MPSImageLanczosScale
- class MPSImageBilinearScale

6.24 MPSImageStatistics.h File Reference

```
#include <MPSImage/MPSImageKernel.h>
#include <simd/simd.h>
```

Classes

- class MPSImageStatisticsMinAndMax
- class MPSImageStatisticsMeanAndVariance
- class MPSImageStatisticsMean

6.25 MPSImageThreshold.h File Reference

```
#include <MPSImage/MPSImageKernel.h>
```

Classes

- class MPSImageThresholdBinary
- class MPSImageThresholdBinaryInverse
- class MPSImageThresholdTruncate
- class MPSImageThresholdToZero
- class MPSImageThresholdToZeroInverse

6.26 MPSImageTranspose.h File Reference

```
#include <MPSImage/MPSImageKernel.h>
```

Classes

- class MPSImageTranspose

6.27 MPSImageTypes.h File Reference

```
#import <MPSCore/MPSCoreTypes.h>
```

Classes

- struct MPSScaleTransform

Typedefs

- typedef enum MPSAlphaType MPSAlphaType
- typedef struct MPSScaleTransform MPSScaleTransform

Enumerations

- enum MPSAlphaType { MPSAlphaTypeNonPremultiplied, MPSAlphaTypeAlphaIsOne, MPSAlphaType←
  Premultiplied }

6.27.1 Typedef Documentation

6.27.1.1 MPSAlphaType

```
typedef enum MPSAlphaType

MPSAlphaType
```

6.27.1.2 MPSScaleTransform

```
typedef struct MPSScaleTransform MPSScaleTransform
```

6.27.2 Enumeration Type Documentation

6.27.2.1 MPSAlphaType

```
enum MPSAlphaType
```

Premultiplication description for the color channels of a texture Some image data is premultiplied. That is to say that the color channels are stored instead as color ∗ alpha. This is an optimization for image compositing (alpha blending), but it can get in the way of most other image filters, especially those that apply non-linear affects like the MPSImageParametricCurveTransform multidimensional lookup tables, and functions like convolution or resampling filters that look at adjacent pixels, where the alpha may not be the same.

```
Some basic conversion cases:
    source                             destination                        operation
    ------                             -----------                        ---------
    MPSAlphaTypeNonPremultiplied       MPSAlphaTypeNonPremultiplied
       <none>
    MPSAlphaTypeNonPremultiplied
      MPSAlphaTypeAlphaIsOne             composite with opaque background color
    MPSAlphaTypeNonPremultiplied
      MPSAlphaTypePremultiplied          multiply color channels by alpha
    MPSAlphaTypeAlphaIsOne
      MPSAlphaTypeNonPremultiplied       set alpha to 1
    MPSAlphaTypeAlphaIsOne
      MPSAlphaTypeAlphaIsOne             set alpha to 1
    MPSAlphaTypeAlphaIsOne
      MPSAlphaTypePremultiplied          set alpha to 1
    MPSAlphaTypePremultiplied
      MPSAlphaTypeNonPremultiplied       divide color channels by alpha
    MPSAlphaTypePremultiplied
      MPSAlphaTypeAlphaIsOne             composite with opaque background color
    MPSAlphaTypePremultiplied          MPSAlphaTypePremultiplied          <none>
```

```
Color space conversion operations require the format to be either MPSPixelAlpha_NonPremultiplied or
MPSPixelAlpha_AlphaIsOne to work correctly. A number of MPSKernels have similar requirements. If
premultiplied data is provided or requested, extra operations will be added to the conversion to
ensure correct operation. Fully opaque images should use MPSAlphaTypeAlphaIsOne.
```

MPSAlphaTypeNonPremultiplied Image is not premultiplied by alpha. Alpha is not guaranteed to be 1. (kCG←ImageAlphaFirst/Last) MPSAlphaTypeAlphaIsOne Alpha is guaranteed to be 1, even if it is not encoded as 1 or not encoded at all. (kCGImageAlphaNoneSkipFirst/Last, kCGImageAlphaNone) MPSAlphaTypePremultiplied Image is premultiplied by alpha. Alpha is not guaranteed to be 1. (kCGImageAlphaPremultipliedFirst/Last)

Enumerator

MPSAlphaTypeNonPremultiplied	
MPSAlphaTypeAlphaIsOne	
MPSAlphaTypePremultiplied	

6.28 MPSKernel.h File Reference

```
#include <MPSCore/MPSCoreTypes.h>
```

Classes

- class MPSKernel

6.29 MPSMatrix.h File Reference

```
#import <MPSMatrix/MPSMatrixTypes.h>
#import <MPSMatrix/MPSMatrixMultiplication.h>
#import <MPSMatrix/MPSMatrixSolve.h>
#import <MPSMatrix/MPSMatrixDecomposition.h>
```

6.30 MPSMatrixDecomposition.h File Reference

```
#import <MPSCore/MPSKernel.h>
#import <MPSMatrix/MPSMatrixTypes.h>
```

Classes

- class MPSMatrixDecompositionLU
- class MPSMatrixDecompositionCholesky

Typedefs

- typedef enum MPSMatrixDecompositionStatus MPSMatrixDecompositionStatus

Enumerations

- enum MPSMatrixDecompositionStatus { MPSMatrixDecompositionStatusSuccess, MPSMatrixDecomposition↩ StatusFailure, MPSMatrixDecompositionStatusSingular, MPSMatrixDecompositionStatusNonPositive↩ Definite }

6.30.1 Typedef Documentation

6.30.1.1 MPSMatrixDecompositionStatus

```
typedef enum MPSMatrixDecompositionStatus
MPSMatrixDecompositionStatus
```

6.30.2 Enumeration Type Documentation

6.30.2.1 MPSMatrixDecompositionStatus

```
enum MPSMatrixDecompositionStatus
```

MPSMatrixDecomposition.h MetalPerformanceShaders.framework

Copyright

 MetalPerformanceShaders filter base classes

A value to indicate the status of a matrix decomposition.

MPSMatrixDecompositionStatusSuccess The decomposition was performed successfully.

MPSMatrixDecompositionStatusFailure The decomposition was not able to be completed.

MPSMatrixDecompositionStatusSingular The resulting decomposition is not suitable for use in a subsequent system solve.

MPSMatrixDecompositionStatusNonPositiveDefinite A non-positive-definite pivot value was calculated.

Enumerator

MPSMatrixDecompositionStatusSuccess	
MPSMatrixDecompositionStatusFailure	
MPSMatrixDecompositionStatusSingular	
MPSMatrixDecompositionStatusNonPositiveDefinite	

6.31 MPSMatrixMultiplication.h File Reference

```
#import <MPSCore/MPSKernel.h>
#import <MPSMatrix/MPSMatrixTypes.h>
```

Classes

- class MPSMatrixMultiplication
- class MPSMatrixVectorMultiplication

6.32 MPSMatrixSolve.h File Reference

```
#import <MPSCore/MPSKernel.h>
#import <MPSMatrix/MPSMatrixTypes.h>
```

Classes

- class MPSMatrixSolveTriangular
- class MPSMatrixSolveLU
- class MPSMatrixSolveCholesky

6.33 MPSMatrixTypes.h File Reference

```
#import <MPSCore/MPSKernel.h>
#import <MPSCore/MPSCoreTypes.h>
```

Classes

- class MPSMatrixDescriptor
- class MPSVectorDescriptor
- class MPSMatrix
- class MPSVector
- class MPSTemporaryMatrix
- class MPSMatrixUnaryKernel
- class MPSMatrixBinaryKernel

6.34 MPSNeuralNetwork.h File Reference

```
#import <MPSNeuralNetwork/MPSNeuralNetworkTypes.h>
#import <MPSNeuralNetwork/MPSCNNKernel.h>
#import <MPSNeuralNetwork/MPSCNNConvolution.h>
#import <MPSNeuralNetwork/MPSCNNPooling.h>
#import <MPSNeuralNetwork/MPSCNNNormalization.h>
#import <MPSNeuralNetwork/MPSCNNSoftMax.h>
#import <MPSNeuralNetwork/MPSCNNUpsampling.h>
#import <MPSNeuralNetwork/MPSRNNLayer.h>
#import <MPSNeuralNetwork/MPSNNGraph.h>
```

6.35 MPSNeuralNetworkTypes.h File Reference

```
#import <MPSCore/MPSCoreTypes.h>
```

Classes

- protocol <MPSNNPadding >
- class MPSNNDefaultPadding
- protocol <MPSImageSizeEncodingState >

Typedefs

- typedef enum MPSCNNConvolutionFlags MPSCNNConvolutionFlags
- typedef enum MPSCNNBinaryConvolutionFlags MPSCNNBinaryConvolutionFlags
- typedef enum MPSCNNBinaryConvolutionType MPSCNNBinaryConvolutionType
- typedef enum MPSNNPaddingMethod MPSNNPaddingMethod

Enumerations

- enum MPSCNNConvolutionFlags { MPSCNNConvolutionFlagsNone }
- enum MPSCNNBinaryConvolutionFlags { MPSCNNBinaryConvolutionFlagsNone, MPSCNNBinaryConvolutionFlagsUseBetaScaling }
- enum MPSCNNBinaryConvolutionType { MPSCNNBinaryConvolutionTypeBinaryWeights, MPSCNNBinaryConvolutionTypeXNOR, MPSCNNBinaryConvolutionTypeAND }
- enum MPSNNPaddingMethod {
  MPSNNPaddingMethodAlignCentered, MPSNNPaddingMethodAlignTopLeft, MPSNNPaddingMethodAlignBottomRight, MPSNNPaddingMethodAlign_reserved,
  MPSNNPaddingMethodAlignMask = MPSNNPaddingMethodAlign_reserved, MPSNNPaddingMethodAddRemainderToTopLeft, MPSNNPaddingMethodAddRemainderToTopRight, MPSNNPaddingMethodAddRemainderToBottomLeft,
  MPSNNPaddingMethodAddRemainderToBottomRight, MPSNNPaddingMethodAddRemainderToMask = MPSNNPaddingMethodAddRemainderToBottomRight, MPSNNPaddingMethodSizeValidOnly, MPSNNPaddingMethodSizeSame,
  MPSNNPaddingMethodSizeFull, MPSNNPaddingMethodSize_reserved, MPSNNPaddingMethodCustom, MPSNNPaddingMethodSizeMask,
  MPSNNPaddingMethodExcludeEdges }

6.35.1 Typedef Documentation

6.35.1.1 MPSCNNBinaryConvolutionFlags

```
typedef enum MPSCNNBinaryConvolutionFlags
MPSCNNBinaryConvolutionFlags
```

6.35.1.2 MPSCNNBinaryConvolutionType

```
typedef enum MPSCNNBinaryConvolutionType
MPSCNNBinaryConvolutionType
```

6.35.1.3 MPSCNNConvolutionFlags

```
typedef enum MPSCNNConvolutionFlags
MPSCNNConvolutionFlags
```

6.35.1.4 MPSNNPaddingMethod

```
typedef enum MPSNNPaddingMethod
MPSNNPaddingMethod
```

6.35.2 Enumeration Type Documentation

6.35.2.1 MPSCNNBinaryConvolutionFlags

```
enum MPSCNNBinaryConvolutionFlags
```

Options used to control CNN Binary convolution kernels.

Enumerator

MPSCNNBinaryConvolutionFlagsNone	Use default in binary convolution options
MPSCNNBinaryConvolutionFlagsUseBetaScaling	Scale the binary convolution operation using the beta-image option as detailed in MPSCNNBinaryConvolution

6.35.2.2 MPSCNNBinaryConvolutionType

```
enum MPSCNNBinaryConvolutionType
```

Defines what operations are used to perform binary convolution

Enumerator

MPSCNNBinaryConvolutionTypeBinaryWeights	Otherwise a normal convolution operation, except that the weights are binary values
MPSCNNBinaryConvolutionTypeXNOR	Use input image binarization and the XNOR-operation to perform the actual convolution - See MPSCNNBinaryConvolution for details
MPSCNNBinaryConvolutionTypeAND	Use input image binarization and the AND-operation to perform the actual convolution - See MPSCNNBinaryConvolution for details

6.35.2.3 MPSCNNConvolutionFlags

```
enum MPSCNNConvolutionFlags
```

Options used to control how kernel weights are stored and used in the CNN kernels. For future expandability.

Enumerator

MPSCNNConvolutionFlagsNone	Use default options

6.35.2.4 MPSNNPaddingMethod

```
enum MPSNNPaddingMethod
```

How to pad MPSNNGraph image nodes The MPSNNGraph must make automatic decisions about how big to make the result of each filter node. This is typically determined by a combination of input image size, size of the filter window (e.g. convolution weights), filter stride, and a description of how much extra space beyond the edges of the image to allow the filter read. By knowing the properties of the filter, we can then infer the size of the result image. Most of this information is known to the MPSNNGraph as part of its normal operation. However, the amount of padding to add and where to add it is a matter of choice left to you, the developer. Different neural network frameworks such as TensorFlow and Caffe make different choices here. Depending on where your network was trained, you will need to adjust the policies used by MPS during inference. In the event that the padding method is not simply described by this enumeration, you may provide you own custom policy definition by overriding the -destinationImageDescriptorForSourceImages: sourceStates:forKernel:suggestedDescriptor: method in a custom MPSNNPadding child class.

Common values that influence the size of the result image by adjusting the amount of padding added to the source images:

- MPSNNPaddingMethodSizeValidOnly Result values are only produced for the area that is guaranteed to have all of its input values defined (i.e. not off the edge). This produces the smallest result image.
- MPSNNPaddingMethodSizeSame The result image is the same size as the input image. If the stride is not 1, then the result is scaled accordingly.
- MPSNNPaddingMethodSizeFull Result values are produced for any position for which at least one input value is defined (i.e. not off the edge)

- MPSNNPaddingMethodCustom The sizing and centering policy is given by the [MPSNNPadding destinationImageDescriptorForSourceImages: sourceStates:forKernel:suggestedDescriptor:]

Except possibly when MPSNNPaddingMethodCustom is used, the area within the source image that is read will be centered on the source image. Even so, at times the area can not be perfectly centered because the source image has odd size and the region read has even size, or vice versa. In such cases, you may use the following values to select where to put the extra padding:

```
- MPSNNPaddingMethodAddRemainderToTopLeft     Leftover padding is added to the top or left
                                              side of image as appropriate.
- MPSNNPaddingMethodAddRemainderToBottomRight Leftover padding is added to the bottom or right
                                              side of image as appropriate.
```

Here again, different external frameworks may use different policies.

In some cases, Caffe intoduces the notion of a region beyond the padding which is invalid. This can happen when the padding is set to a width narrower than what is needed for a destination size. In such cases, MPSNNPadding←MethodExcludeEdges is used to adjust normalization factors for filter weights (particularly in pooling) such that invalid regions beyond the padding are not counted towards the filter area. Currently, only pooling supports this feature. Other filters ignore it.

The MPSNNPaddingMethodSize and a MPSNNPaddingMethodAddRemainder policy always appear together in the MPSNNPaddingMethod. There is no provision for a MPSNNPaddingMethodSize without a remainder policy or vice versa. It is in practice used as a bit field.

Most MPSNN filters are considered forward filters. Some (e.g. convolution transpose and unpooling) are considered reverse filters. For the reverse filters, the image stride is measured in destination values rather than source values and has the effect of enlarging the image rather than reducing it. When a reverse filter is used to "undo" the effects of a forward filter, the MPSNNPaddingMethodSize should be the opposite of the forward MPSNNPadding←Method. For example, if the forward filter used MPSNNPaddingMethodSizeValidOnly | MPSNNPaddingMethod←AddRemainderToTopLeft, the reverse filter should use MPSNNPaddingMethodSizeFull | MPSNNPaddingMethod←AddRemainderToTopLeft. Some consideration of the geometry of inputs and outputs will reveal why this is so. It is usually not important to adjust the centering method because the size of the reverse result generally doesn't suffer from centering asymmetries. That is: the size would usually be given by:

```
static int DestSizeReverse( int sourceSize, int stride, int filterWindowSize, Style style ) {
    return (sourceSize-1) * stride + 1 + style  * (filterWindowSize-1);  // style = {-1,0,1} for
       valid-only, same, full
}
```

so the result size is exactly the one needed for the source size and there are no centering problems. In some cases where the reverse pass is intended to completely reverse a forward pass, the MPSState object produced by the forward pass should be used to determine the size of the reverse pass result image.

Tensorflow does not appear to provide a full padding method, but instead appears to use its valid-only padding mode for reverse filters to in effect achieve what is called MPSNNPaddingMethodSizeFull here.

MPSGetPaddingPolicy() is provided as a convenience to make shorter work of MPSNNPaddingMethods and policies.

Walkthrough of operation of padding policy:

Most MPSCNNKernels have two types of -encode calls. There is one for which you must pass in a preallocated MPSImage to receive the results. This is for manual configuration. It assumes you know what you are doing, and asks you to correctly set a diversity of properties to correctly position image inputs and size results. It does not use the padding policy. You must size the result correctly, set the clipRect, offset and other properties as needed yourself. Layered on top of that is usually another flavor of -encode call that returns a destination image instead from the left hand side of the function. It is designed to automatically configure itself based on the MPSCNNKernel.↩ paddingPolicy. When this more automated -encode... method is called, it invokes a method in the MPSKernel that looks at the MPSNNPaddingMethod bitfield of the policy. Based on the information therein and the size of the input images and other filter properties, it determines the size of the output, sets the offset property, and returns an appropriate MPSImageDescriptor for the destination image. If you set the MPSNNPaddingMethodCustom bit in the MPSNNPaddingMethod, then the MPSNNPadding -destinationImageDescriptorForSourceImages:sourceStates↩ :forKernel:suggestedDescriptor: method is called. The MPSImageDescriptor prepared earlier is passed in as the last parameter. You can use this descriptor or modify as needed. In addition, you can adjust any properties of the MPSKernel with which it will be used. If, for example, the descriptor is not the right MPSFeatureChannelFormat, you can change it, or make your own MPSImageDescriptor based on the one handed to you. This is your opportunity to customize the configuration of the MPSKernel. In some cases (e.g. paddingForTensorflowAveragePooling (MPS↩ NNDefaultPadding) you might change other properties such as the filter edging mode, or adjust the offset that was already set for you. When the kernel is fully configured, return the MPSImageDescriptor. The MPSImageDescriptor is then passed to the MPSCNNKernel.destinationImageAllocator to allocate the image. You might provide such an allocator if you want to use your own custom MTLHeap rather than the MPS internal heap. The allocator can be set either directly in the MPSCNNKernel or through the MPSNNImageNode.allocator property. It is intended that most of the time, default values for padding method and destination image allocator should be good enough. Only minimal additional configuration should be required, apart from occasional adjustments to set the MPSNNPaddingMethod when something other than default padding for the object is needed. If you find yourself encumbered by frequent adjustments of this kind, you might find it to your advantage to subclass MPSNNFilterNodes or MPSCNNKernels to adjust the default padding policy and allocator at initialization time.

```
tensorFlowSame = MPSNNPaddingMethodAddRemainderToBottomRight | MPSNNPaddingMethodAlignCentered | MPSNN
```

Enumerator

MPSNNPaddingMethodAlignCentered	
MPSNNPaddingMethodAlignTopLeft	
MPSNNPaddingMethodAlignBottomRight	
MPSNNPaddingMethodAlign_reserved	
MPSNNPaddingMethodAlignMask	
MPSNNPaddingMethodAddRemainderToTopLeft	
MPSNNPaddingMethodAddRemainderToTopRight	
MPSNNPaddingMethodAddRemainderToBottomLeft	
MPSNNPaddingMethodAddRemainderToBottomRight	
MPSNNPaddingMethodAddRemainderToMask	
MPSNNPaddingMethodSizeValidOnly	
MPSNNPaddingMethodSizeSame	
MPSNNPaddingMethodSizeFull	
MPSNNPaddingMethodSize_reserved	
MPSNNPaddingMethodCustom	
MPSNNPaddingMethodSizeMask	
MPSNNPaddingMethodExcludeEdges	The caffe framework constrains the average pooling area to the limits of the padding area in cases where a pixel would read beyond the padding area. Set this bit for Caffe emulation with average pooling.

6.36 MPSNNGraph.h File Reference

```
#include <MPSNeuralNetwork/MPSNNGraphNodes.h>
#include <MPSCore/MPSKernel.h>
```

Classes

- class MPSNNGraph

Typedefs

- typedef void(^ MPSNNGraphCompletionHandler) (MPSImage ∗__nullable result, NSError ∗__nullable error)

6.36.1 Typedef Documentation

6.36.1.1 MPSNNGraphCompletionHandler

```
typedef void(^ MPSNNGraphCompletionHandler) (MPSImage *__nullable result, NSError *__nullable
error)
```

A notification when computeAsyncWithSourceImages:completionHandler: has finished

Parameters

result	If no error, the image produced by the graph operation.
error	If an error occurs, more information might be found here.

6.37 MPSNNGraphNodes.h File Reference

```
#include <MPSCore/MPSImage.h>
#include <MPSCore/MPSState.h>
#include <MPSNeuralNetwork/MPSNeuralNetworkTypes.h>
#include <MPSNeuralNetwork/MPSCNNNeuronType.h>
```

Classes

- protocol <MPSHandle >
- class MPSNNImageNode
- class MPSNNStateNode
- class MPSCNNConvolutionStateNode

- class MPSNNFilterNode
- class MPSCNNConvolutionNode
- class MPSCNNFullyConnectedNode
- class MPSCNNBinaryConvolutionNode
- class MPSCNNBinaryFullyConnectedNode
- class MPSCNNConvolutionTransposeNode
- class MPSCNNNeuronNode
- class MPSCNNNeuronAbsoluteNode
- class MPSCNNNeuronELUNode
- class MPSCNNNeuronLinearNode
- class MPSCNNNeuronReLUNode
- class MPSCNNNeuronSigmoidNode
- class MPSCNNNeuronHardSigmoidNode
- class MPSCNNNeuronSoftPlusNode
- class MPSCNNNeuronSoftSignNode
- class MPSCNNNeuronTanHNode
- class MPSCNNPoolingNode
- class MPSCNNPoolingAverageNode
- class MPSCNNPoolingL2NormNode
- class MPSCNNPoolingMaxNode
- class MPSCNNDilatedPoolingMaxNode
- class MPSCNNNormalizationNode
- class MPSCNNSpatialNormalizationNode
- class MPSCNNLocalContrastNormalizationNode
- class MPSCNNCrossChannelNormalizationNode
- class MPSNNBinaryArithmeticNode
- class MPSNNAdditionNode
- class MPSNNSubtractionNode
- class MPSNNMultiplicationNode
- class MPSNNDivisionNode
- class MPSNNConcatenationNode
- class MPSCNNSoftMaxNode
- class MPSCNNLogSoftMaxNode
- class MPSCNNUpsamplingNearestNode
- class MPSCNNUpsamplingBilinearNode

6.38 MPSRNNLayer.h File Reference

```
#include <MPSNeuralNetwork/MPSCNNConvolution.h>
#include <MPSMatrix/MPSMatrix.h>
```

Classes

- class MPSRNNDescriptor
- class MPSRNNSingleGateDescriptor
- class MPSGRUDescriptor
- class MPSLSTMDescriptor
- class MPSRNNRecurrentImageState
- class MPSRNNImageInferenceLayer
- class MPSRNNRecurrentMatrixState
- class MPSRNNMatrixInferenceLayer

Typedefs

- typedef enum MPSRNNSequenceDirection MPSRNNSequenceDirection
- typedef enum MPSRNNBidirectionalCombineMode MPSRNNBidirectionalCombineMode

Enumerations

- enum MPSRNNSequenceDirection { MPSRNNSequenceDirectionForward, MPSRNNSequenceDirection↩ Backward }
- enum MPSRNNBidirectionalCombineMode { MPSRNNBidirectionalCombineModeNone, MPSRNN↩ BidirectionalCombineModeAdd, MPSRNNBidirectionalCombineModeConcatenate }

6.38.1 Typedef Documentation

6.38.1.1 MPSRNNBidirectionalCombineMode

```
typedef enum MPSRNNBidirectionalCombineMode

MPSRNNBidirectionalCombineMode
```

6.38.1.2 MPSRNNSequenceDirection

```
typedef enum MPSRNNSequenceDirection

MPSRNNSequenceDirection
```

6.38.2 Enumeration Type Documentation

6.38.2.1 MPSRNNBidirectionalCombineMode

```
enum MPSRNNBidirectionalCombineMode
```

Defines the way in which two images or matrices are combined together, or if the results are to be kept separate.

See also

MPSRNNImageInferenceLayer and
MPSRNNMatrixInferenceLayer.

Enumerator

MPSRNNBidirectionalCombineModeNone	The two sequences are kept separate
MPSRNNBidirectionalCombineModeAdd	The two sequences are summed together to form a single output
MPSRNNBidirectionalCombineModeConcatenate	The two sequences are concatenated together along the feature channels to form a single output

6.38.2.2 MPSRNNSequenceDirection

```
enum MPSRNNSequenceDirection
```

Defines the direction in which a sequence of inputs is processed by a RNN Layer.

See also

MPSRNNImageInferenceLayer and
MPSRNNMatrixInferenceLayer.

Enumerator

MPSRNNSequenceDirectionForward	The input sequence is processed from index zero to array length minus one
MPSRNNSequenceDirectionBackward	The input sequence is processed from index array length minus one to zero

6.39 MPSState.h File Reference

```
#import <MPSCore/MPSCoreTypes.h>
```

Classes

- class MPSState

Index