Using Metal 2 for Compute
Session 608

Anna Tikhonova, GPU Software Engineer
Metal 2 Ecosystem

Metal API and language
GPU Tools
MetalKit
Metal Performance Shaders
Metal 2 Ecosystem

Metal API and language
GPU Tools
MetalKit
Metal Performance Shaders

Metal 2
Metal Performance Shaders (MPS)

GPU accelerated primitives
• Image Processing
• Linear Algebra
• Machine Learning — Inference

Optimized for iOS
# Metal Performance Shaders (MPS)

GPU accelerated primitives

- Image Processing
- Linear Algebra
- Machine Learning — Inference

Optimized for iOS and macOS

<table>
<thead>
<tr>
<th>What's New in Metal, Part 2</th>
<th>WWDC 2016</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>WWDC 2015</td>
</tr>
</tbody>
</table>
Image Processing
## Image Processing
Primitives available in iOS 10

<table>
<thead>
<tr>
<th>Convolution</th>
<th>Equalization and Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gaussian Blur</td>
<td>Median</td>
</tr>
<tr>
<td>Box, Tent</td>
<td>Thresholding</td>
</tr>
<tr>
<td>Sobel</td>
<td>Transpose</td>
</tr>
<tr>
<td>Morphology</td>
<td>Image Integral</td>
</tr>
<tr>
<td>Lanczos Resampling</td>
<td>Color Conversion</td>
</tr>
<tr>
<td>Histogram</td>
<td>Gaussian Pyramid</td>
</tr>
</tbody>
</table>
Image Processing
New primitives

Image Keypoints
Bilinear Rescale
Image Statistics
Element-wise Arithmetic Operations
• With broadcasting
Linear Algebra

New primitives

Matrix-Matrix Multiplication

Matrix-Vector Multiplication

Triangular Matrix Factorization and Linear Solvers
Data Representations

MPSVector

- Interprets data in MTLBuffer as a 1-dimensional array
Data Representations

MPSVector
• Interprets data in MTLBuffer as a 1-dimensional array

MPSMatrix
• Interprets data in MTLBuffer as a rectangular array
• Row-major order
Data Representations

MPSVector
• Interprets data in MTLBuffer as a 1-dimensional array

MPSMatrix
• Interprets data in MTLBuffer as a rectangular array
• Row-major order

MPSTemporaryMatrix
• Allocated from MTLHeap
• Use for most of your intermediate matrices
MPSVector and MPSMatrix

Input types

Single Precision Floating-Point
Half Precision Floating-Point
16-bit Signed Integer
8-bit Signed Integer
Create a vector of size N

```swift
// Create a Metal buffer of length N
let buffer = device.makeBuffer(length: N * MemoryLayout<Float32>.size)

// Create a vector descriptor
let descriptor = MPSVectorDescriptor(length: N, dataType: .float32)

// Create a vector with descriptor
let vector = MPSVector(buffer: buffer, descriptor: descriptor)
```
Create a vector of size N

```swift
// Create a Metal buffer of length N
let buffer = device.makeBuffer(length: N * MemoryLayout<Float32>.size)

// Create a vector descriptor
let descriptor = MPSVectorDescriptor(length: N, dataType: .float32)

// Create a vector with descriptor
let vector = MPSVector(buffer: buffer, descriptor: descriptor)
```
Create a vector of size N

```swift
// Create a Metal buffer of length N
let buffer = device.makeBuffer(length: N * MemoryLayout<Float32>.size)

// Create a vector descriptor
let descriptor = MPSVectorDescriptor(length: N, dataType: .float32)

// Create a vector with descriptor
let vector = MPSVector(buffer: buffer, descriptor: descriptor)
```
Create a vector of size N

// Create a Metal buffer of length N
let buffer = device.makeBuffer(length: N * MemoryLayout<Float32>.size)

// Create a vector descriptor
let descriptor = MPSVectorDescriptor(length: N, dataType: .float32)

// Create a vector with descriptor
let vector = MPSVector(buffer: buffer, descriptor: descriptor)
Create a matrix with M rows and N columns

// Get the recommended bytes per row value to use for sizing a Metal buffer
let bytesPerRow = MPSMatrixDescriptor.rowBytes(forColumns: N, dataType: .float32)

// Create a Metal buffer with the recommended bytes per row
let buffer = device.makeBuffer(length: M * bytesPerRow)

// Create a matrix descriptor
let descriptor = MPSMatrixDescriptor(rows: M, columns: N, rowBytes: bytesPerRow, dataType: .float32)

// Create a matrix with descriptor
let matrix = MPSMatrix(buffer: buffer, descriptor: descriptor)
Create a matrix with \( M \) rows and \( N \) columns

```swift
// Get the recommended bytes per row value to use for sizing a Metal buffer
let bytesPerRow = MPSMatrixDescriptor.rowBytes(forColumns: N, dataType: .float32)

// Create a Metal buffer with the recommended bytes per row
let buffer = device.makeBuffer(length: M * bytesPerRow)

// Create a matrix descriptor
let descriptor = MPSMatrixDescriptor(rows: M, columns: N, rowBytes: bytesPerRow, dataType: .float32)

// Create a matrix with descriptor
let matrix = MPSMatrix(buffer: buffer, descriptor: descriptor)
```
Create a matrix with M rows and N columns

```swift
// Get the recommended bytes per row value to use for sizing a Metal buffer
let bytesPerRow = MPSMatrixDescriptor.rowBytes(forColumns: N, dataType: .float32)

// Create a Metal buffer with the recommended bytes per row
let buffer = device.makeBuffer(length: M * bytesPerRow)

// Create a matrix descriptor
let descriptor = MPSMatrixDescriptor(rows: M, columns: N, rowBytes: bytesPerRow, dataType: .float32)

// Create a matrix with descriptor
let matrix = MPSMatrix(buffer: buffer, descriptor: descriptor)
```
Get the recommended bytes per row value to use for sizing a Metal buffer

```swift
let bytesPerRow = MPSMatrixDescriptor.rowBytes(forColumns: N, dataType: .float32)
```

Create a Metal buffer with the recommended bytes per row

```swift
let buffer = device.makeBuffer(length: M * bytesPerRow)
```

Create a matrix descriptor

```swift
let descriptor = MPSMatrixDescriptor(rows: M, columns: N, rowBytes: bytesPerRow, 
                                  dataType: .float32)
```

Create a matrix with descriptor

```swift
let matrix = MPSMatrix(buffer: buffer, descriptor: descriptor)
```
Primitives

Matrix-Matrix and Matrix-Vector Multiplication
• API modeled after standard BLAS GEMM and GEMV interfaces

Triangular Matrix Factorization and Linear Solvers
• API modeled after standard LAPACK decomposition and solve interfaces
// Example: Matrix-Matrix Multiply: C = A B

// Create matrices A, B and C
let A = MPSMatrix(buffer: ABuffer,
    descriptor: MPSMatrixDescriptor(rows: M, columns: K,
        rowBytes: ARowBytes, dataType: .float32))

let B = MPSMatrix(buffer: BBuffer,
    descriptor: MPSMatrixDescriptor(rows: K, columns: N,
        rowBytes: BRowBytes, dataType: .float32))

let C = MPSMatrix(buffer: CBuffer,
    descriptor: MPSMatrixDescriptor(rows: M, columns: N,
        rowBytes: CRowBytes, dataType: .float32))
// Example: Matrix–Matrix Multiply: C = A B

// Perform Metal setup
let device = MTLCreateSystemDefaultDevice()!
let commandQueue = device.makeCommandQueue()
let commandBuffer = commandQueue.makeCommandBuffer()

// Create a Matrix–Matrix Multiplication kernel
let mmKernel = MPSMatrixMultiplication(device: device, resultRows: M,
                                         resultColumns: N, interiorColumns: K)

// Encode kernel to the command buffer
mmKernel.encode(commandBuffer: commandBuffer, leftMatrix: A,
                 rightMatrix: B, resultMatrix: C)

// Tell GPU to start doing the work
commandBuffer.commit()
// Example: Matrix–Matrix Multiply: C = A B

// Perform Metal setup
let device = MTLCreateSystemDefaultDevice()!
let commandQueue = device.makeCommandQueue()
let commandBuffer = commandQueue.makeCommandBuffer()

// Create a Matrix–Matrix Multiplication kernel
let mmKernel = MPSMatrixMultiplication(device: device, resultRows: M,
                                        resultColumns: N, interiorColumns: K)

// Encode kernel to the command buffer
mmKernel.encode(commandBuffer: commandBuffer, leftMatrix: A,
                 rightMatrix: B, resultMatrix: C)

// Tell GPU to start doing the work
commandBuffer.commit()
// Example: Matrix-Matrix Multiply: C = A B

// Perform Metal setup
let device = MTLCreateSystemDefaultDevice()!
let commandQueue = device.makeCommandQueue()
let commandBuffer = commandQueue.makeCommandBuffer()

// Create a Matrix-Matrix Multiplication kernel
let mmKernel = MPSMatrixMultiplication(device: device, resultRows: M,
                                        resultColumns: N, interiorColumns: K)

// Encode kernel to the command buffer
mmKernel.encode(commandBuffer: commandBuffer, leftMatrix: A,
                 rightMatrix: B, resultMatrix: C)

// Tell GPU to start doing the work
commandBuffer.commit()
// Example: Matrix-Matrix Multiply: C = A B

// Perform Metal setup
let device = MTLCreateSystemDefaultDevice()!
let commandQueue = device.makeCommandQueue()
let commandBuffer = commandQueue.makeCommandBuffer()

// Create a Matrix-Matrix Multiplication kernel
let mmKernel = MPSMatrixMultiplication(device: device, resultRows: M,
                                        resultColumns: N, interiorColumns: K)

// Encode kernel to the command buffer
mmKernel.encode(commandBuffer: commandBuffer, leftMatrix: A,
                rightMatrix: B, resultMatrix: C)

// Tell GPU to start doing the work
commandBuffer.commit()
Sample Code

MPSMatrixMultiplication
https://developer.apple.com/library/content/samplecode/MPSMatrixMultiplicationSample

Triangular Matrix Factorization and Linear Solvers
Coming soon
Machine Learning
Machine Learning at Apple

Architecture

Domain Specific Frameworks
- Vision
- NLP

ML Framework
- Core ML

ML Performance Primitives
- Accelerate
- MPS
What Is Deep Learning?
Training and Inference

Training to Classify Images

- cat
- rabbit
- dog
- giraffe
- horse
Training to Classify Images
Training to Classify Images

- cat
- rabbit
- dog
- giraffe
- horse
Training to Classify Images

- cat
- rabbit
- dog
- giraffe
- horse

Trained Parameters
Inference

Training to Classify Images

- cat
- rabbit
- dog
- giraffe
- horse

Trained Parameters
Inference

Training to Classify Images

Input Image

CNN

Inference

cat
Agenda

Recap on Convolutional Neural Networks (CNN)
Agenda

Recap on Convolutional Neural Networks (CNN)
Convolutional Neural Networks — New Primitives
Neural Network Graph API
Recurrent Neural Networks (RNN)
Agenda

Recap on Convolutional Neural Networks (CNN)
Convolutional Neural Networks — New Primitives
Neural Network Graph API
Recurrent Neural Networks (RNN)
What Are Convolutional Neural Networks?
Convolutional Neural Networks

Biologically-inspired, resemble the visual cortex
Convolutional Neural Networks

Biologically-inspired, resemble the visual cortex

Hierarchical representation
• Organized into a hierarchy of layers
• Higher-level features are derived from lower-level features
Convolutional Neural Networks

Biologically-inspired, resemble the visual cortex

Hierarchical representation
• Organized into a hierarchy of layers
• Higher-level features are derived from lower-level features

Think of a "feature" as a filter that filters data for that feature
# Convolutional Neural Networks

Primitives available in iOS 10

<table>
<thead>
<tr>
<th>Convolution</th>
<th>Fully-Connected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pooling</td>
<td>Neuron</td>
</tr>
<tr>
<td>• Average</td>
<td>• Linear</td>
</tr>
<tr>
<td>• Max</td>
<td>• ReLU</td>
</tr>
<tr>
<td>Normalization</td>
<td>• Sigmoid</td>
</tr>
<tr>
<td>• Cross-Channel</td>
<td>• TanH</td>
</tr>
<tr>
<td>• Local Contrast</td>
<td>• Absolute</td>
</tr>
<tr>
<td>• Spatial</td>
<td></td>
</tr>
</tbody>
</table>
Convolutional Neural Networks
Primitives available in iOS 10

Convolution

Pooling
• Average
• Max

Normalization
• Cross-Channel
• Local Contrast
• Spatial

Fully-Connected

Softmax

Neuron
• Linear
• ReLU
• Sigmoid
• TanH
• Absolute
Convolution

Core building block

Recognizes features in input
1-channel output

1-channel input

1 filter
3 x 3
1-channel output

1 filter
3 x 3

1-channel input

1-channel output
1-channel input

1 filter
3 x 3

1-channel output
1-channel output

1 filter
3 x 3

1-channel input

1-channel output
1-channel output

1 filter
3 x 3

1-channel input

1-channel output
3-channel input
40 x 40

3*16 5x5 filters

16-channel output
40 x 40
3-channel input
40 x 40

3*16 5x5 filters

16-channel output
40 x 40
3-channel input
40 x 40

3*16 5x5 filters

16-channel output
40 x 40
Agenda

Recap on Convolutional Neural Networks (CNN)
Convolutional Neural Networks — New Primitives
Neural Network Graph API
Recurrent Neural Networks (RNN)
Convolutional Neural Networks

New primitives

New Convolution weight types
Binary and XNOR Convolution
Sub-Pixel Convolution
Dilated Convolution
Convolution Transpose
L2Norm Pooling
Dilated Max Pooling
Log Softmax

Resampling
• Lanczos, Bilinear

Upsampling

Arithmetic Operators
• Addition, Subtraction, Multiplication, Division

New Neuron layers
• Hard Sigmoid, SoftPlus, SoftSign, ELU
Convolutional Neural Networks

New primitives

New Convolution weight types
Binary and XNOR Convolution
Sub-Pixel Convolution
Dilated Convolution
Convolution Transpose
L2Norm Pooling
Dilated Max Pooling
Log Softmax

Resampling
• Lanczos, Bilinear

Upsampling

Arithmetic Operators
• Addition, Subtraction, Multiplication, Division

New Neuron layers
• Hard Sigmoid, SoftPlus, SoftSign, ELU
Convolution
Filter weight types

Single Precision Floating-Point
To reduce memory footprint and improve performance
• Half Precision Floating-Point
• 8-bit Integer
• Binary
Convolution
Primitives

Standard

Binary and XNOR

Dilated

Sub-Pixel

Transpose
Binary and XNOR Convolution

Same operation as regular Convolution

Improved performance

Less memory
Binary Convolution

- Full-sized input, binary weights

### Binary and XNOR Convolution

**Regular Convolution**

**Input**

<table>
<thead>
<tr>
<th>0.15</th>
<th>0.12</th>
<th>0.23</th>
<th>0.31</th>
<th>-0.51</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.16</td>
<td>-0.70</td>
<td>0.85</td>
<td>-0.67</td>
<td>0.79</td>
</tr>
<tr>
<td>0.73</td>
<td>0.92</td>
<td>-0.63</td>
<td>0.72</td>
<td>-0.11</td>
</tr>
<tr>
<td>0.86</td>
<td>-0.66</td>
<td>-0.12</td>
<td>0.19</td>
<td>0.23</td>
</tr>
</tbody>
</table>

**Weights**

<table>
<thead>
<tr>
<th>0.65</th>
<th>-0.78</th>
<th>-0.36</th>
<th>0.72</th>
<th>0.13</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.68</td>
<td>-0.09</td>
<td>0.21</td>
<td>-0.39</td>
<td>0.23</td>
</tr>
</tbody>
</table>

**Binary Convolution**

**Input**

<table>
<thead>
<tr>
<th>0.15</th>
<th>0.12</th>
<th>0.23</th>
<th>0.31</th>
<th>-0.51</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.16</td>
<td>-0.70</td>
<td>0.85</td>
<td>-0.67</td>
<td>0.79</td>
</tr>
<tr>
<td>0.73</td>
<td>0.92</td>
<td>-0.63</td>
<td>0.72</td>
<td>-0.11</td>
</tr>
<tr>
<td>0.86</td>
<td>-0.66</td>
<td>-0.12</td>
<td>0.19</td>
<td>0.23</td>
</tr>
</tbody>
</table>

**Weights**

<table>
<thead>
<tr>
<th>+1</th>
<th>-1</th>
<th>-1</th>
<th>+1</th>
<th>+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>-1</td>
<td>+1</td>
<td>-1</td>
<td>+1</td>
</tr>
<tr>
<td>+1</td>
<td>-1</td>
<td>+1</td>
<td>-1</td>
<td>+1</td>
</tr>
</tbody>
</table>
Binary and XNOR Convolution

Binary Convolution
• Full-sized input, binary weights

XNOR Convolution
• Binary input, binary weights

Regular Convolution

Binary Convolution

XNOR Convolution

Input

Weights

$0.15 \ 0.12 \ 0.23 \ 0.31 \ -0.51$

$0.16 \ -0.70 \ 0.85 \ -0.67 \ 0.79$

$0.73 \ 0.92 \ -0.63 \ 0.72 \ -0.11$

$0.86 \ -0.66 \ -0.12 \ 0.19 \ 0.23$

$0.65 \ -0.78 \ -0.36 \ 0.72 \ 0.13$

$0.68 \ -0.09 \ 0.21 \ -0.39 \ 0.23$

$+1 \ -1 \ -1 \ +1 \ +1$

$+1 \ -1 \ +1 \ -1 \ +1$

$+1 \ -1 \ +1 \ -1 \ +1$

$+1 \ -1 \ -1 \ +1 \ +1$

$+1 \ -1 \ -1 \ +1 \ +1$
Dilated Convolution
Comparison to regular convolution

Input

Output
Dilated Convolution
Comparison to regular convolution
Dilated Convolution
Comparison to regular convolution

Input

Output

3 x 3 kernel
Dilated Convolution
Comparison to regular convolution

Input

Output

3 x 3 kernel
Dilated Convolution

How it works

Input

Output

3 x 3 kernel
dilationFactorX = 2
dilationFactorY = 2
Dilated Convolution

How it works

Input

Output

3 x 3 kernel

dilationFactorX = 2
dilationFactorY = 2
Sub-Pixel Convolution and Convolution Transpose

Commonly used for upscaling
Upscaling
Using a box filter

Fixed operation with a constant filter

Input
W x H

Output
2W x 2H
Upscaling
Using a box filter

Fixed operation with a constant filter

Input
W x H

Output
2W x 2H
Upscaling
Using a box filter

Fixed operation with a constant filter

Input
$W \times H$

Output
$2W \times 2H$
Sub-Pixel Convolution

How it works

One-channel input  
$W \times H$

Trained Parameters

4 filters  
for 2x upscaling

One-channel output  
$2W \times 2H$
Sub-Pixel Convolution

How it works

One-channel input
W x H

4 filters
for 2x upscaling

One-channel output
2W x 2H
Sub-Pixel Convolution

How it works

One-channel input
W x H

4 filters
for 2x upscaling

One-channel output
2W x 2H

Reshuffle
Convolution Transpose

How it works

Input
W x H
Convolution Transpose
How it works

Input
W x H
Convolution Transpose

How it works

Intermediate Result
2W x 2H

Output
W x H
Convolution Transpose

How it works

Intermediate Result
2W x 2H

Output
W x H
Convolution Transpose

How it works

Intermediate Result
2W x 2H

Output
W x H
Convolution Transpose

How it works

Intermediate Result
2W x 2H

Output
W x H
Convolution Transpose

How it works

Intermediate Result
2W x 2H

Output
W x H
Convolution Transpose

How it works

Intermediate Result
2W x 2H

Output
W x H
New Convolution Primitives
Example: colorizing black and white images
New Convolution Primitives
Example: colorizing black and white images

New Convolution Primitives
Example: colorizing black and white images

- Dilated Convolution—integrate wider global context

New Convolution Primitives

Example: colorizing black and white images

- Dilated Convolution—integrate wider global context
- Convolution Transpose—upscale output

Demo

Image colorization
Performance Improvements in iOS 11

Higher is better

Percentage Improvement

0 20 40

iPhone 6S  iPhone 7 Plus  iPad Pro 9.7”  iPad Pro 10.5”

Inception-v3 network

Performance Improvements in iOS 11

Higher is better

Percentage Improvement

- iPhone 6S: 22%
- iPhone 7 Plus: 22%
- iPad Pro 9.7"": 29%
- iPad Pro 10.5"": 21%

Agenda

Recap on Convolutional Neural Networks (CNN)
Convolutional Neural Networks — New Primitives
Neural Network Graph API
Recurrent Neural Networks (RNN)
Neural Network Graph API

Overview

Describe neural network using graph API
Neural Network Graph API
Overview

Describe neural network using graph API
Describe neural network using graph API

Neural Network Graph API
Overview

Convolution
Pooling (Avg.)
Pooling (Max.)
Fully-Connected
SoftMax
Concatenation
Image
Neural Network Graph API
Overview

Describe neural network using graph API

Filter nodes — Operations

- Convolution
- Pooling (Avg.)
- Pooling (Max.)
- Fully-Connected
- SoftMax
- Concatentation
- Image
Neural Network Graph API

Overview

Describe neural network using graph API

Filter nodes — Operations

Image nodes — Data

Convolution
Pooling (Avg.)
Pooling (Max.)
Fully-Connected
SoftMax
Concatenation
Image
Neural Network Graph API

Ease of use

Compact representation
Neural Network Graph API
Ease of use

Compact representation

Save and restore across platforms (NSSecureCoding)
Neural Network Graph API
Ease of use

Compact representation

Save and restore across platforms (NSSecureCoding)

Initialize once, reuse
Neural Network Graph API
Ease of use

Compact representation

Save and restore across platforms (NSSecureCoding)

Initialize once, reuse

Execute graph on GPU with single call
Neural Network Graph API

Ease of use

Compact representation

Save and restore across platforms (NSSecureCoding)

Initialize once, reuse

Execute graph on GPU with single call

No intermediate images to manage, just input/output
Neural Network Graph API

Ease of use

Compact representation

Save and restore across platforms (NSSecureCoding)

Initialize once, reuse

Execute graph on GPU with single call

No intermediate images to manage, just input/output

Auto-configuration of image sizes, padding, centering
Neural Network Graph API

Ease of use

Compact representation

Save and restore across platforms (NSSecureCoding)

Initialize once, reuse

Execute graph on GPU with single call

No intermediate images to manage, just input/output

Auto-configuration of image sizes, padding, centering

MetallImageRecognition code sample* — 4x less code with NN Graph API

https://developer.apple.com/library/content/samplecode/MetallImageRecognition
Neural Network Graph API
Deliver best performance

Easy to parallelize between CPU and GPU
Neural Network Graph API
Deliver best performance

Easy to parallelize between CPU and GPU
Fuse graph nodes
Neural Network Graph API
Deliver best performance

Easy to parallelize between CPU and GPU

Fuse graph nodes

Execute graph nodes concurrently
Neural Network Graph API
Deliver best performance

Easy to parallelize between CPU and GPU
Fuse graph nodes
Execute graph nodes concurrently
Neural Network Graph API
Deliver best performance

Easy to parallelize between CPU and GPU
Fuse graph nodes
Execute graph nodes concurrently
Optimize away Concatenation nodes
Neural Network Graph API
Deliver best performance

Easy to parallelize between CPU and GPU
Fuse graph nodes
Execute graph nodes concurrently
Optimize away Concatenation nodes
Filter Nodes
Convolution node

Create a MPSNNConvolutionNode with data source provider

```swift
let conv1 = MPSNNConvolutionNode(source: MPSNNImageNode(handle: nil),
                                   weights: MyWeights(file: "conv1.dat"))
```
Create a MPSNNConvolutionNode with data source provider

```swift
let conv1 = MPSNNConvolutionNode(source: MPSNNImageNode(handle: nil),
                                  weights: MyWeights(file: "conv1.dat"))
```
Create a MPSNNConvolutionNode with data source provider

```
let conv1 = MPSNNConvolutionNode(source: MPSNNImageNode(handle: nil),
                                 weights: MyWeights(file: "conv1.dat"))
```
Feeding Parameters to Convolution Layer

Just-in-time loading and purging of weights data

Minimize memory footprint

class MyWeights: NSObject, MPSCNNConvolutionDataSource {
    // Initialize the data source object
    init(file: String) {...}

    public func load() -> Bool {...}
    public func descriptor() -> MPSCNNConvolutionDescriptor {...}
    public func weights() -> UnsafeMutableRawPointer {...}
    public func purge() {...
}
Feeding Parameters to Convolution Layer

Just-in-time loading and purging of weights data

Minimize memory footprint

class MyWeights: NSObject, MPSCNNConvolutionDataSource {
    // Initialize the data source object
    init(file: String) {...}

    public func load() -> Bool {...}
    public func descriptor() -> MPSCNNConvolutionDescriptor {...}
    public func weights() -> UnsafeMutableRawPointer {...}
    public func purge() {...}
}
Feeding Parameters to Convolution Layer

Just-in-time loading and purging of weights data

Minimize memory footprint

class MyWeights: NSObject, MPSCNNConvolutionDataSource {
    // Initialize the data source object
    init(file: String) {...}

    public func load() -> Bool {...}
    public func descriptor() -> MPSCNNConvolutionDescriptor {...}
    public func weights() -> UnsafeMutableRawPointer {...}
    public func purge() {...}
}
// Example: create a graph
func makeGraph() -> MPSNNImageNode {
    conv1
    pool1
    conv2
    pool2
    conv3
    pool3
    conv4
    fc1
    fc2
}
// Example: create a graph

func makeGraph() -> MPSNNImageNode {

    let conv1 = MPSCNNConvolutionNode(source: MPSNNImageNode(handle: nil), weights: MyWeights(file:"conv1.dat"))

    ...
func makeGraph() -> MPSNNImageNode {

    let conv1 = MPSCNNConvolutionNode(source: MPSNNImageNode(handle: nil), weights: MyWeights(file:"conv1.dat"))

    let pool1 = MPSCNNPoolingMaxNode(source: conv1.resultImage, filterSize: 2)

    return MPSNNImageNode(handle: nil)
}
// Example: create a graph
func makeGraph() -> MPSNNImageNode {

    let conv1 = MPSCNNConvolutionNode(source: MPSNNImageNode(handle: nil), weights: MyWeights(file:"conv1.dat"))
    let pool1 = MPSCNNPoolingMaxNode(source: conv1.resultImage, filterSize: 2)
    let conv2 = MPSCNNConvolutionNode(source: pool1.resultImage, weights: MyWeights(file:"conv2.dat"))
    let pool2 = MPSCNNPoolingMaxNode(source: conv2.resultImage, filterSize: 2)
    let conv3 = MPSCNNConvolutionNode(source: pool2.resultImage, weights: MyWeights(file:"conv3.dat"))
    let pool3 = MPSCNNPoolingMaxNode(source: conv3.resultImage, filterSize: 2)
    let conv4 = MPSCNNConvolutionNode(source: pool3.resultImage, weights: MyWeights(file:"conv4.dat"))
    let fc1 = MPSCNNFullyConnectedNode(source: conv4.resultImage, weights: MyWeights(file:"fc1.dat"))
    let fc2 = MPSCNNFullyConnectedNode(source: fc1.resultImage, weights: MyWeights(file:"fc2.dat"))
    return fc2.resultImage
}
// Example: create a graph
func makeGraph() -> MPSNNImageNode {

    let conv1 = MPSCNNConvolutionNode(source: MPSNNImageNode(handle: nil), weights: MyWeights(file:"conv1.dat"))
    let pool1 = MPSCNNPoolingMaxNode(source: conv1.resultImage, filterSize: 2)
    let conv2 = MPSCNNConvolutionNode(source: pool1.resultImage, weights: MyWeights(file:"conv2.dat"))
    let pool2 = MPSCNNPoolingMaxNode(source: conv2.resultImage, filterSize: 2)
    let conv3 = MPSCNNConvolutionNode(source: pool2.resultImage, weights: MyWeights(file:"conv3.dat"))
    let pool3 = MPSCNNPoolingMaxNode(source: conv3.resultImage, filterSize: 2)
    let conv4 = MPSCNNConvolutionNode(source: pool3.resultImage, weights: MyWeights(file:"conv4.dat"))
    let fc1 = MPSCNNFullyConnectedNode(source: conv4.resultImage, weights: MyWeights(file:"fc1.dat"))
    let fc2 = MPSCNNFullyConnectedNode(source: fc1.resultImage, weights: MyWeights(file:"fc2.dat"))

    return fc2.resultImage
}
// Example: execute graph on the GPU

// Metal setup
let device = MTLCreateSystemDefaultDevice()!
let commandQueue = device.makeCommandQueue()
let commandBuffer = commandQueue.makeCommandBuffer()

// Initialize graph
let graph = MPSNNGraph(device: device, resultImage: makeGraph())

// Create input image
let input = MPSImage(texture: texture, …)

// Encode graph
let output = graph?.encode(to: commandBuffer,
    sourceImages: [input])

// Tell GPU to start executing work and wait until GPU work is done
commandBuffer.commit()
commandBuffer.waitUntilCompleted()
// Example: execute graph on the GPU

// Metal setup
let device = MTLCreateSystemDefaultDevice()!
let commandQueue = device.makeCommandQueue()
let commandBuffer = commandQueue.makeCommandBuffer()

// Initialize graph
let graph = MPSNNGraph(device: device, resultImage: makeGraph())

// Create input image
let input = MPSImage(texture: texture, …)

// Encode graph
let output = graph?.encode(to: commandBuffer,
    sourceImages: [input])

// Tell GPU to start executing work and wait until GPU work is done
commandBuffer.commit()
commandBuffer.waitUntilCompleted()
// Example: execute graph on the GPU

// Metal setup
let device = MTLCreateSystemDefaultDevice()!
let commandQueue = device.makeCommandQueue()
let commandBuffer = commandQueue.makeCommandBuffer()

// Initialize graph
let graph = MPSNNGraph(device: device, resultImage: makeGraph())

// Create input image
let input = MPSImage(texture: texture, …)

// Encode graph
let output = graph?.encode(to: commandBuffer,
                          sourceImages: [input])

// Tell GPU to start executing work and wait until GPU work is done
commandBuffer.commit()
commandBuffer.waitUntilCompleted()
// Example: execute graph on the GPU

// Metal setup
let device = MTLCreateSystemDefaultDevice()!
let commandQueue = device.makeCommandQueue()
let commandBuffer = commandQueue.makeCommandBuffer()

// Initialize graph
let graph = MPSNNGraph(device: device, resultImage: makeGraph())

// Create input image
let input = MPSImage(texture: texture, ...)

// Encode graph
let output = graph?.encode(to: commandBuffer,
    sourceImages: [input])

// Tell GPU to start executing work and wait until GPU work is done
commandBuffer.commit()
commandBuffer.waitUntilCompleted()
// Example: execute graph on the GPU
// Metal setup
let device = MTLCreateSystemDefaultDevice()!
let commandQueue = device.makeCommandQueue()
let commandBuffer = commandQueue.makeCommandBuffer()

// Initialize graph
let graph = MPSNNGraph(device: device, resultImage: makeGraph())

// Create input image
let input = MPSImage(texture: texture, …)

// Encode graph
let output = graph?.encode(to: commandBuffer,
sourceImages: [input])

// Tell GPU to start executing work and wait until GPU work is done
commandBuffer.commit()
commandBuffer.waitUntilCompleted()
// Example: execute graph on the GPU
// Metal setup
let device = MTLCreateSystemDefaultDevice()!
let commandQueue = device.makeCommandQueue()
let commandBuffer = commandQueue.makeCommandBuffer()

// Initialize graph
let graph = MPSNNGraph(device: device, resultImage: makeGraph())

// Create input image
let input = MPSImage(texture: texture, …)

// Encode graph
let output = graph?.encode(to: commandBuffer,
                      sourceImages: [input])

// Tell GPU to start executing work and wait until GPU work is done
commandBuffer.commit()
commandBuffer.waitUntilCompleted()
// Example: execute graph on the GPU
// Metal setup
let device = MTLCreateSystemDefaultDevice()!
let commandQueue = device.makeCommandQueue()
let commandBuffer = commandQueue.makeCommandBuffer()

// Initialize graph
let graph = MPSNNGraph(device: device, resultImage: makeGraph())

// Create input image
let input = MPSImage(texture: texture, …)

// Encode graph
let output = graph?.encode(to: commandBuffer,
    sourceImages: [input])

// Tell GPU to start executing work and wait until GPU work is done
commandBuffer.commit()
commandBuffer.waitUntilCompleted()
// Example: execute graph on the GPU

// Metal setup
let device = MTLCreateSystemDefaultDevice()!
let commandQueue = device.makeCommandQueue()
let commandBuffer = commandQueue.makeCommandBuffer()

// Initialize graph
let graph = MPSNNGraph(device: device, resultImage: makeGraph())

// Create input image
let input = MPSImage(texture: texture, …)

// Encode graph
let output = graph?.encode(to: commandBuffer,
sourceImages: [input])

// Tell GPU to start executing work and wait until GPU work is done
commandBuffer.commit()
commandBuffer.waitUntilCompleted()
// Example: execute graph on the GPU asynchronously

// Metal setup
let device = MTLCreateSystemDefaultDevice()!

// Initialize graph
let graph = MPSNNGraph(device: device, resultImage: makeGraph())

// Create input image
let input = MPSImage(texture: texture, ...

// Encode graph
let output = graph?.executeAsync(sourceImages: [input]) {
    resultImage, error in
        // check for error and use resultImage inside closure
}

// Don't wait, encode new GPU task
// Example: execute graph on the GPU asynchronously

// Metal setup
let device = MTLCreateSystemDefaultDevice()!

// Initialize graph
let graph = MPSNNGraph(device: device, resultImage: makeGraph())

// Create input image
let input = MPSImage(texture: texture, …)

// Encode graph
let output = graph?.executeAsync(sourceImages: [input]) {
    resultImage, error in
    // check for error and use resultImage inside closure
}

// Don’t wait, encode new GPU task
// Example: execute graph on the GPU asynchronously

// Metal setup
let device = MTLCreateSystemDefaultDevice()!

// Initialize graph
let graph = MPSNNGraph(device: device, resultImage: makeGraph())

// Create input image
let input = MPSImage(texture: texture, …)

// Encode graph
let output = graph?.executeAsync(sourceImages: [input]) {
    resultImage, error in
    // check for error and use resultImage inside closure
}

// Don’t wait, encode new GPU task
// Example: execute graph on the GPU asynchronously

// Metal setup
let device = MTLCreateSystemDefaultDevice()!

// Initialize graph
let graph = MPSNNGraph(device: device, resultImage: makeGraph())

// Create input image
let input = MPSImage(texture: texture, …)

// Encode graph
let output = graph?.executeAsync(sourceImages: [input]) {
    resultImage, error in
    // check for error and use resultImage inside closure
}

// Don’t wait, encode new GPU task
// Example: execute graph on the GPU asynchronously

// Metal setup
let device = MTLCreateSystemDefaultDevice()!

// Initialize graph
let graph = MPSNNGraph(device: device, resultImage: makeGraph())

// Create input image
let input = MPSImage(texture: texture, …)

// Encode graph
let output = graph?.executeAsync(sourceImages: [input]) { resultImage, error in
  // check for error and use resultImage inside closure
}

// Don't wait, encode new GPU task
let device = MTLCreateSystemDefaultDevice()!

let graph = MPSNNGraph(device: device, resultImage: makeGraph())

let input = MPSImage(texture: texture, …)

let output = graph?.executeAsync(sourceImages: [input]) {
    resultImage, error in
    // check for error and use resultImage inside closure
}

// Don’t wait, encode new GPU task
Demo

Inception-v3 using Neural Network Graph API
Agenda

- Recap on Convolutional Neural Networks (CNN)
- Convolutional Neural Networks — New Primitives
- Neural Network Graph API
- Recurrent Neural Networks (RNN)
What Are Recurrent Neural Networks?
CNN
One-to-one

One input
Image
CNN
One-to-one

Inference

One input
Image

One output
Set of probabilities

dog
grass
...
RNN
Sequences: one-to-many
RNN
Sequences: one-to-many

A black and white dog laying in the grass

One input
Set of probabilities

Sequence of outputs
Words / image caption
RNN
Sequences: many - to - many

A black and white dog laying in the grass
A black and white dog laying in the grass.

Inference

Чёрно-белая собака лежит на траве.

Mustan ja valkisen värinen koira makaa ruohikolla.

Sequence of inputs
Sentence in English

Sequence of outputs
Translated sentence
Recurrent Neural Networks
New primitives

Single Gate
Long Short-Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Minimally Gated Unit (MGU)
Recurrent Unit enables previous output to affect the output of subsequent iterations.
Built from Single Gate RNNs

Has an internal Memory Cell

Gates control information flow inside the LSTM and what is stored in the Memory Cell
Long Short-Term Memory (LSTM)

- Built from Single Gate RNNs
- Has an internal Memory Cell
- Gates control information flow inside the LSTM and what is stored in the Memory Cell
LSTM
Architecture

Output

Input

LSTM
Memory Cell
LSTM
Architecture
LSTM Architecture

Old Memory

New Memory
LSTM Architecture

- Old Memory
- New Memory
- Previous Output
- Input
- Forget Gate

What to keep from old memory

Matrix-Matrix or Matrix-Vector Multiply
Point-wise operations

\[ M \]
\[ * \]
\[ + \]
LSTM

Architecture

Old Memory → LSTM → New Memory

Old Memory: What to keep from old memory
New Memory: How new input affects new memory

Input Gates, Forget Gates, and Cell Gates:
- Input Gates: Matrix-Matrix or Matrix-Vector Multiply
- Forget Gates: Point-wise operations

+ Matrix-Matrix or Matrix-Vector Multiply
* Point-wise operations

How new input affects new memory
What to keep from old memory
LSTM Architecture

- How new input affects new memory
- What to keep from old memory
- Matrix-Matrix or Matrix-Vector Multiply
- Point-wise operations

Old Memory

- Forget Gate

New Memory

- Input Gate

Cell Gate

- Previous Output
- Input

Matrix-Matrix or Matrix-Vector Multiply
Point-wise operations
What to keep from old memory
How new input affects new memory

Matrix-Matrix or Matrix-Vector Multiply
Point-wise operations
LSTM Architecture

- **Old Memory**
- **New Memory**
- **Input**
- **Output**

**Gates:**
- **Forget Gate**
- **Input Gate**
- **Cell Gate**

**Operations:**
- Matrix-Matrix or Matrix-Vector Multiply
- Point-wise operations
- What to keep from old memory
- How new input affects new memory

How new input affects new memory
What to keep from old memory
Matrix-Matrix or Matrix-Vector Multiply
Point-wise operations
LSTM Architecture

- Output Gate
- Forget Gate
- Input Gate
- Cell Gate

- How new input affects new memory
- How previous output, current input, new memory affect new output
- What to keep from old memory

- Matrix-Matrix or Matrix-Vector Multiply
- Point-wise operations

- Previous Output
- Input
- Old Memory
- New Memory

- Matrix-Matrix or Matrix-Vector Multiply
- Point-wise operations

- *
// Example: Creating a LSTM RNN

// Create a LSTM layer descriptor
let descriptor = MPSLSTMDescr iptor()
descriptor.inputFeatureChannels = inputSize
descriptor.outputFeatureChannels = outputSize

// Create and initialize gate weights with trained parameters, using a data source provider
// for just-in-time loading and purging of weights
descriptor.forgetGateInputWeights = MyWeights(file: "forgetGateWeights.dat")
descriptor.cellGateInputWeights = MyWeights(file: "cellGateWeights.dat")

// Initialize the rest of the gates…

// Metal setup
let device = MTLCreateSystemDefaultDevice()! // Also get commandQueue and commandBuffer

// Create a LSTM layer
let layer = MPSRNNMatrixInferenceLayer(device: device, rnnDescriptor: descriptor)
// Example: Creating a LSTM RNN

// Create a LSTM layer descriptor
let descriptor = MPSLSTMDescr iptor()
descriptor.inputFeatureChannels = inputSize
descriptor.outputFeatureChannels = outputSize

// Create and initialize gate weights with trained parameters, using a data source provider
// for just-in-time loading and purging of weights
descriptor.forgetGateInputWeights = MyWeights(file:"forgetGateWeights.dat")
descriptor.cellGateInputWeights = MyWeights(file:"cellGateWeights.dat")

// Initialize the rest of the gates...

// Metal setup
let device = MTLCreateSystemDefaultDevice()! // Also get commandQueue and commandBuffer

// Create a LSTM layer
let layer = MPSRNNMatrixInferenceLayer(device: device, rnn Descriptor: descriptor)
/* Example: Creating a LSTM RNN */

/* Create a LSTM layer descriptor */
let descriptor = MPSLSTMDesciptor()
descriptor.inputFeatureChannels = inputSize
descriptor.outputFeatureChannels = outputSize

/* Create and initialize gate weights with trained parameters, using a data source provider */
/* for just-in-time loading and purging of weights */
descriptor.forgetGateInputWeights = MyWeights(file:"forgetGateWeights.dat")
descriptor.cellGateInputWeights = MyWeights(file:"cellGateWeights.dat")
/* Initialize the rest of the gates... */

/* Metal setup */
let device = MTLCreateSystemDefaultDevice()! // Also get commandQueue and commandBuffer

/* Create a LSTM layer */
let layer = MPSRNNMatrixInferenceLayer(device: device, rnnDescriptor: descriptor)
// Example: Creating a LSTM RNN

// Create a LSTM layer descriptor
let descriptor = MPSLSTMDescriptor()
descriptor.inputFeatureChannels = inputSize
descriptor.outputFeatureChannels = outputSize

// Create and initialize gate weights with trained parameters, using a data source provider
// for just-in-time loading and purging of weights
descriptor.forgetGateInputWeights = MyWeights(file: "forgetGateWeights.dat")
descriptor.cellGateInputWeights = MyWeights(file: "cellGateWeights.dat"))
// Initialize the rest of the gates...

// Metal setup
let device = MTLCreateSystemDefaultDevice()! // Also get commandQueue and commandBuffer

// Create a LSTM layer
let layer = MPSRNNMatrixInferenceLayer(device: device, rnnDescriptor: descriptor)
// Example: Running a LSTM RNN on the GPU

// Create input and output data
var inputSequence: [MPSMatrix] = []
var outputSequence: [MPSMatrix] = []
for i in 0..< N {
    // Matrix size is (1, inputSize), inputSize is number of columns
    inputSequence.append(MPSMatrix(...))
    // Matrix size is (1, outputSize), outputSize is number of columns
    outputSequence.append(MPSMatrix(...))
}

// Submit work to GPU
layer.encodeSequence(commandBuffer: commandBuffer,
    sourceMatrices: inputSequence, destinationMatrices: outputSequence,
    recurrentInputState: nil, recurrentOutputStates: nil)

// Tell GPU to start executing work
commandBuffer.commit()
// Example: Running a LSTM RNN on the GPU

// Create input and output data
var inputSequence: [MPSMatrix] = []
var outputSequence: [MPSMatrix] = []
for i in 0..< N {
    // Matrix size is (1, inputSize), inputSize is number of columns
    inputSequence.append(MPSMatrix(...))
    // Matrix size is (1, outputSize), outputSize is number of columns
    outputSequence.append(MPSMatrix(...))
}

// Submit work to GPU
layer.encodeSequence(commandBuffer: commandBuffer,
    sourceMatrices: inputSequence, destinationMatrices: outputSequence,
    recurrentInputState: nil, recurrentOutputStates: nil)

// Tell GPU to start executing work
commandBuffer.commit()
// Example: Running a LSTM RNN on the GPU

// Create input and output data
var inputSequence: [MPSMatrix] = []
var outputSequence: [MPSMatrix] = []
for i in 0..< N {
    // Matrix size is (1, inputSize), inputSize is number of columns
    inputSequence.append(MPSMatrix(…))
    // Matrix size is (1, outputSize), outputSize is number of columns
    outputSequence.append(MPSMatrix(…))
}

// Submit work to GPU
layer.encodeSequence(commandBuffer: commandBuffer,
                      sourceMatrices: inputSequence, destinationMatrices: outputSequence,
                      recurrentInputState: nil, recurrentOutputStates: nil)

// Tell GPU to start executing work
commandBuffer.commit()
Example: Image Captioning

Training to Caption Images
Example: Image Captioning

Training to Caption Images

Training

Trained Parameters
Example: Image Captioning

Training

CNN

Determine what is depicted in the image

RNN

Generate image caption

Trained Parameters
Example: Image Captioning

Inference

Trained Parameters
Example: Image Captioning

Inference

CNN

Determine what is depicted in the image

RNN

Generate image caption

Trained Parameters

- \text{CNN}
  - Determine what is depicted in the image
- \text{RNN}
  - Generate image caption
Example: Image Captioning

Inference

Determine what is depicted in the image

Generate image caption

CNN

RNN

Trained Parameters control CNN layers

Trained Parameters control RNN gates
Example: Image Captioning

Inference

Determine what is depicted in the image

Generate image caption

CNN

RNN
Example: Image Captioning

Inference

Determine what is depicted in the image

Generate image caption

CNN

RNN

a man riding a wave on top of a surfboard
Example: Image Captioning

Inference

Determine what is depicted in the image

Inception-v3

Generate image caption

LSTM

Memory Cell

Image Captioning Network*

a man riding a wave on top of a surfboard

Example: Image Captioning

LSTM initialization phase

Inception-v3

Convolution
Pooling (Avg.)
Pooling (Max.)
Fully-Connected
SoftMax

LSTM Memory Cell
Example: Image Captioning

LSTM initialization phase

Inception-v3

LSTM

Memory Cell

Convolution
Pooling (Avg.)
Pooling (Max.)
Fully-Connected
SoftMax
Example: Image Captioning
LSTM initialization phase

Inception-v3

Convolution
Pooling (Avg.)
Pooling (Max.)
Fully-Connected
SoftMax

Feature vector

LSTM
Memory Cell
Example: Image Captioning
LSTM initialization phase

Inception-v3

Feature vector

LSTM
Memory Cell

Convolution
Pooling (Avg.)
Pooling (Max.)
Fully-Connected
SoftMax
Example: Image Captioning
Caption generation phase

Input
Sentence start token

LSTM
Memory Cell

Output

Convolution
Pooling (Avg.)
Pooling (Max.)
Fully-Connected
SoftMax
Example: Image Captioning
Caption generation phase

- Input
- Sentence start token
- LSTM
- Memory Cell
- Output
- 3 best one-word captions

- Convolution
- Pooling (Avg.)
- Pooling (Max.)
- Fully-Connected
- SoftMax
Example: Image Captioning
Caption generation phase

Input
- Sentence start token
- LSTM Memory Cell
- 3 best one-word captions

Output
- 3 best one-word captions
- LSTM Memory Cell
- 3 best two-word captions

Symbols: Convolution, Pooling (Avg.), Pooling (Max.), Fully-Connected, SoftMax
Example: Image Captioning

Caption generation phase

Input
Sentence start token
LSTM Memory Cell
Output
3 best one-word captions
End
3 best two-word captions
3 best N-word captions

Convolution
Pooling (Avg.)
Pooling (Max.)
Fully-Connected
SoftMax
Caption Generation

Top three captions:
1. 
2. 
3. 

<table>
<thead>
<tr>
<th>Caption</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>man</td>
<td>0.021986</td>
</tr>
<tr>
<td>a</td>
<td>0.862899</td>
</tr>
<tr>
<td>the</td>
<td>0.039906</td>
</tr>
</tbody>
</table>

Iteration 2

<table>
<thead>
<tr>
<th>Caption</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Caption Generation

Top three captions:
1. 
2. 
3. 

<table>
<thead>
<tr>
<th>Caption</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>man</td>
<td>0.021986</td>
</tr>
<tr>
<td>a</td>
<td>0.862899</td>
</tr>
<tr>
<td>the</td>
<td>0.039906</td>
</tr>
</tbody>
</table>

Iteration 2

<table>
<thead>
<tr>
<th>Caption</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Caption Generation

Top three captions:
1. a
2. the
3. man

<table>
<thead>
<tr>
<th>Caption</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>man</td>
<td>0.021986</td>
</tr>
<tr>
<td>a</td>
<td>0.862899</td>
</tr>
<tr>
<td>the</td>
<td>0.039906</td>
</tr>
</tbody>
</table>

**Iteration 2**

<table>
<thead>
<tr>
<th>Caption</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>man on</td>
<td>0.005290</td>
</tr>
<tr>
<td>man in</td>
<td>0.004869</td>
</tr>
<tr>
<td>man surfing</td>
<td>0.003914</td>
</tr>
</tbody>
</table>
Caption Generation

Top three captions:
1. a man
2. the
3. a

Iteration 1

<table>
<thead>
<tr>
<th>Caption</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>man</td>
<td>0.021986</td>
</tr>
<tr>
<td>a</td>
<td>0.862899</td>
</tr>
<tr>
<td>the</td>
<td>0.039906</td>
</tr>
</tbody>
</table>

Iteration 2

<table>
<thead>
<tr>
<th>Caption</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>man on</td>
<td>0.005290</td>
</tr>
<tr>
<td>man in</td>
<td>0.004869</td>
</tr>
<tr>
<td>man surfing</td>
<td>0.003914</td>
</tr>
<tr>
<td>a man</td>
<td>0.385814</td>
</tr>
<tr>
<td>a person</td>
<td>0.136590</td>
</tr>
<tr>
<td>a surfer</td>
<td>0.116651</td>
</tr>
</tbody>
</table>
### Top three captions:
1. a
2. the
3.

### Iteration 1

<table>
<thead>
<tr>
<th>Caption</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>man</td>
<td>0.021986</td>
</tr>
<tr>
<td>a</td>
<td>0.862899</td>
</tr>
<tr>
<td>the</td>
<td>0.039906</td>
</tr>
</tbody>
</table>

### Iteration 2

<table>
<thead>
<tr>
<th>Caption</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>man on</td>
<td>0.005290</td>
</tr>
<tr>
<td>man in</td>
<td>0.004869</td>
</tr>
<tr>
<td>man surfing</td>
<td>0.003914</td>
</tr>
<tr>
<td>a man</td>
<td>0.385814</td>
</tr>
<tr>
<td>a person</td>
<td>0.136590</td>
</tr>
<tr>
<td>a surfer</td>
<td>0.116651</td>
</tr>
<tr>
<td>the man</td>
<td>0.014275</td>
</tr>
<tr>
<td>the surfer</td>
<td>0.012315</td>
</tr>
<tr>
<td>the young</td>
<td>0.003500</td>
</tr>
</tbody>
</table>
## Caption Generation

Top three captions:
1. 
2. 
3.

### Iteration 1

<table>
<thead>
<tr>
<th>Caption</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>man</td>
<td>0.021986</td>
</tr>
<tr>
<td>a</td>
<td>0.862899</td>
</tr>
<tr>
<td>the</td>
<td>0.039906</td>
</tr>
</tbody>
</table>

### Iteration 2

<table>
<thead>
<tr>
<th>Caption</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>man on</td>
<td>0.005290</td>
</tr>
<tr>
<td>man in</td>
<td>0.004869</td>
</tr>
<tr>
<td>man surfing</td>
<td>0.003914</td>
</tr>
<tr>
<td>a man</td>
<td>0.385814</td>
</tr>
<tr>
<td>a person</td>
<td>0.136590</td>
</tr>
<tr>
<td>a surfer</td>
<td>0.116651</td>
</tr>
<tr>
<td>the man</td>
<td>0.014275</td>
</tr>
<tr>
<td>the surfer</td>
<td>0.012315</td>
</tr>
<tr>
<td>the young</td>
<td>0.003500</td>
</tr>
</tbody>
</table>
## Caption Generation

**Top three captions:**

1. [Caption](#)
2. [Caption](#)
3. [Caption](#)

| Iteration 2 | | Iteration 3 | |
|-------------|-----------------|-----------------|
| **Caption** | **Probability** | **Caption** | **Probability** |
| man on      | 0.005290        | a man riding    | 0.115423        |
| man in      | 0.004869        | a man on        | 0.060142        |
| man surfing | 0.003914        | a man is        | 0.048678        |
| a man       | 0.385814        | a person riding | 0.041114        |
| a person    | 0.136590        | a person on     | 0.031153        |
| a surfer    | 0.116651        | a person in     | 0.014218        |
| the man     | 0.014275        | a surfer is     | 0.027462        |
| the surfer  | 0.012315        | a surfer riding | 0.016631        |
| the young   | 0.003500        | a surfer in     | 0.015737        |
Caption Generation

Top three captions:
1.
2.
3.

Iteration 2

<table>
<thead>
<tr>
<th>Caption</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>man on</td>
<td>0.005290</td>
</tr>
<tr>
<td>man in</td>
<td>0.004869</td>
</tr>
<tr>
<td>man surfing</td>
<td>0.003914</td>
</tr>
<tr>
<td>a man</td>
<td>0.385814</td>
</tr>
<tr>
<td>a person</td>
<td>0.136590</td>
</tr>
<tr>
<td>a surfer</td>
<td>0.116651</td>
</tr>
<tr>
<td>the man</td>
<td>0.014275</td>
</tr>
<tr>
<td>the surfer</td>
<td>0.012315</td>
</tr>
<tr>
<td>the young</td>
<td>0.003500</td>
</tr>
</tbody>
</table>

Iteration 3

<table>
<thead>
<tr>
<th>Caption</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>a man riding</td>
<td>0.115423</td>
</tr>
<tr>
<td>a man on</td>
<td>0.060142</td>
</tr>
<tr>
<td>a man is</td>
<td>0.048678</td>
</tr>
<tr>
<td>a person riding</td>
<td>0.041114</td>
</tr>
<tr>
<td>a person on</td>
<td>0.031153</td>
</tr>
<tr>
<td>a person in</td>
<td>0.014218</td>
</tr>
<tr>
<td>a surfer is</td>
<td>0.027462</td>
</tr>
<tr>
<td>a surfer riding</td>
<td>0.016631</td>
</tr>
<tr>
<td>a surfer in</td>
<td>0.015737</td>
</tr>
</tbody>
</table>
## Caption Generation

### Iteration 3

<table>
<thead>
<tr>
<th>Caption</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>a man riding</td>
<td>0.115423</td>
</tr>
<tr>
<td>a man on</td>
<td>0.060142</td>
</tr>
<tr>
<td>a man is</td>
<td>0.048678</td>
</tr>
<tr>
<td>a person riding</td>
<td>0.041114</td>
</tr>
<tr>
<td>a person on</td>
<td>0.031153</td>
</tr>
<tr>
<td>a person in</td>
<td>0.014218</td>
</tr>
<tr>
<td>a surfer is</td>
<td>0.027462</td>
</tr>
<tr>
<td>a surfer riding</td>
<td>0.016631</td>
</tr>
<tr>
<td>a surfer in</td>
<td>0.015737</td>
</tr>
</tbody>
</table>

### Iteration 4

<table>
<thead>
<tr>
<th>Caption</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>a man riding a</td>
<td>0.100079</td>
</tr>
<tr>
<td>a man riding on</td>
<td>0.008264</td>
</tr>
<tr>
<td>a man riding the</td>
<td>0.002604</td>
</tr>
<tr>
<td>a man on a</td>
<td>0.055997</td>
</tr>
<tr>
<td>a man on his</td>
<td>0.001288</td>
</tr>
<tr>
<td>a man on the</td>
<td>0.001211</td>
</tr>
<tr>
<td>a surfer is surfing</td>
<td>0.027462</td>
</tr>
<tr>
<td>a man is riding</td>
<td>0.016631</td>
</tr>
<tr>
<td>a man is on</td>
<td>0.015737</td>
</tr>
<tr>
<td>a man is on</td>
<td>0.003434</td>
</tr>
</tbody>
</table>
### Caption Generation

**Top three captions:**
1. a man riding
2. a man on
3. a man is

#### Iteration 3

<table>
<thead>
<tr>
<th>Caption</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>a man riding</td>
<td>0.115423</td>
</tr>
<tr>
<td>a man on</td>
<td>0.060142</td>
</tr>
<tr>
<td>a man is</td>
<td>0.048678</td>
</tr>
<tr>
<td>a person riding</td>
<td>0.041114</td>
</tr>
<tr>
<td>a person on</td>
<td>0.031153</td>
</tr>
<tr>
<td>a person in</td>
<td>0.014218</td>
</tr>
<tr>
<td>a surfer is</td>
<td>0.027462</td>
</tr>
<tr>
<td>a surfer riding</td>
<td>0.016631</td>
</tr>
<tr>
<td>a surfer in</td>
<td>0.015737</td>
</tr>
</tbody>
</table>

#### Iteration 4

<table>
<thead>
<tr>
<th>Caption</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>a man riding a</td>
<td>0.100079</td>
</tr>
<tr>
<td>a man riding on</td>
<td>0.008264</td>
</tr>
<tr>
<td>a man riding the</td>
<td>0.002604</td>
</tr>
<tr>
<td>a man on a</td>
<td>0.055997</td>
</tr>
<tr>
<td>a man on his</td>
<td>0.001288</td>
</tr>
<tr>
<td>a man on the</td>
<td>0.001211</td>
</tr>
<tr>
<td>a surfer is</td>
<td>0.021393</td>
</tr>
<tr>
<td>a surfer riding</td>
<td>0.015222</td>
</tr>
<tr>
<td>a surfer in</td>
<td>0.003434</td>
</tr>
</tbody>
</table>
Caption Generation

Top three captions:
1. a man riding a wave on top of a surfboard
2. a man on a surfboard riding a wave
3. a man riding a wave on a surfboard
Caption Generation

Top three captions:
1. a man riding a wave on top of a surfboard
2. a man on a surfboard riding a wave
3. a man riding a wave on a surfboard
Demo
Image captioning — CNN + LSTM
Summary

GPU accelerated primitives
  • Expanded support for Image Processing and Convolutional Neural Networks
  • Added support for Linear Algebra and Recurrent Neural Networks

Optimized for iOS and macOS

New Neural Network Graph API
<table>
<thead>
<tr>
<th>Session</th>
<th>Location</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introducing Metal 2</td>
<td>Executive Ballroom</td>
<td>Tuesday 1:50PM</td>
</tr>
<tr>
<td>Introducing Core ML</td>
<td>Hall 3</td>
<td>Tuesday 3:10PM</td>
</tr>
<tr>
<td>VR with Metal 2</td>
<td>Hall 3</td>
<td>Wednesday 10:00AM</td>
</tr>
<tr>
<td>Vision Framework: Building on Core ML</td>
<td>Hall 2</td>
<td>Wednesday 3:10PM</td>
</tr>
<tr>
<td>Core ML in depth</td>
<td>Hall 3</td>
<td>Thursday 09:00AM</td>
</tr>
<tr>
<td>Accelerate and Sparse Solvers</td>
<td>Executive Ballroom</td>
<td>Thursday 10:00AM</td>
</tr>
<tr>
<td>Metal 2 Optimization and Debugging</td>
<td>Grand Ballroom B</td>
<td>Thursday 3:10PM</td>
</tr>
<tr>
<td>Labs</td>
<td>Metal 2 Lab</td>
<td>Technology Lab</td>
</tr>
</tbody>
</table>
More Information
