#WWDC18

Metal for Accelerating Machine Learning

Session 609

Anna Tikhonova, GPU Software Engineer

Metal Performance Shaders

GPU-accelerated primitives, optimized for iOS and macOS

- Image processing
- Linear algebra
- Machine learning—inference

Using Metal 2 for Compute	WWDC 2017
What’s New in Metal, Part 2	WWDC 2016

Metal Performance Shaders

NEW

GPU-accelerated primitives, optimized for iOS and macOS

- Image processing
- Linear algebra
- Machine learning—inference and training

Metal Performance Shaders

NEW

GPU-accelerated primitives, optimized for iOS and macOS

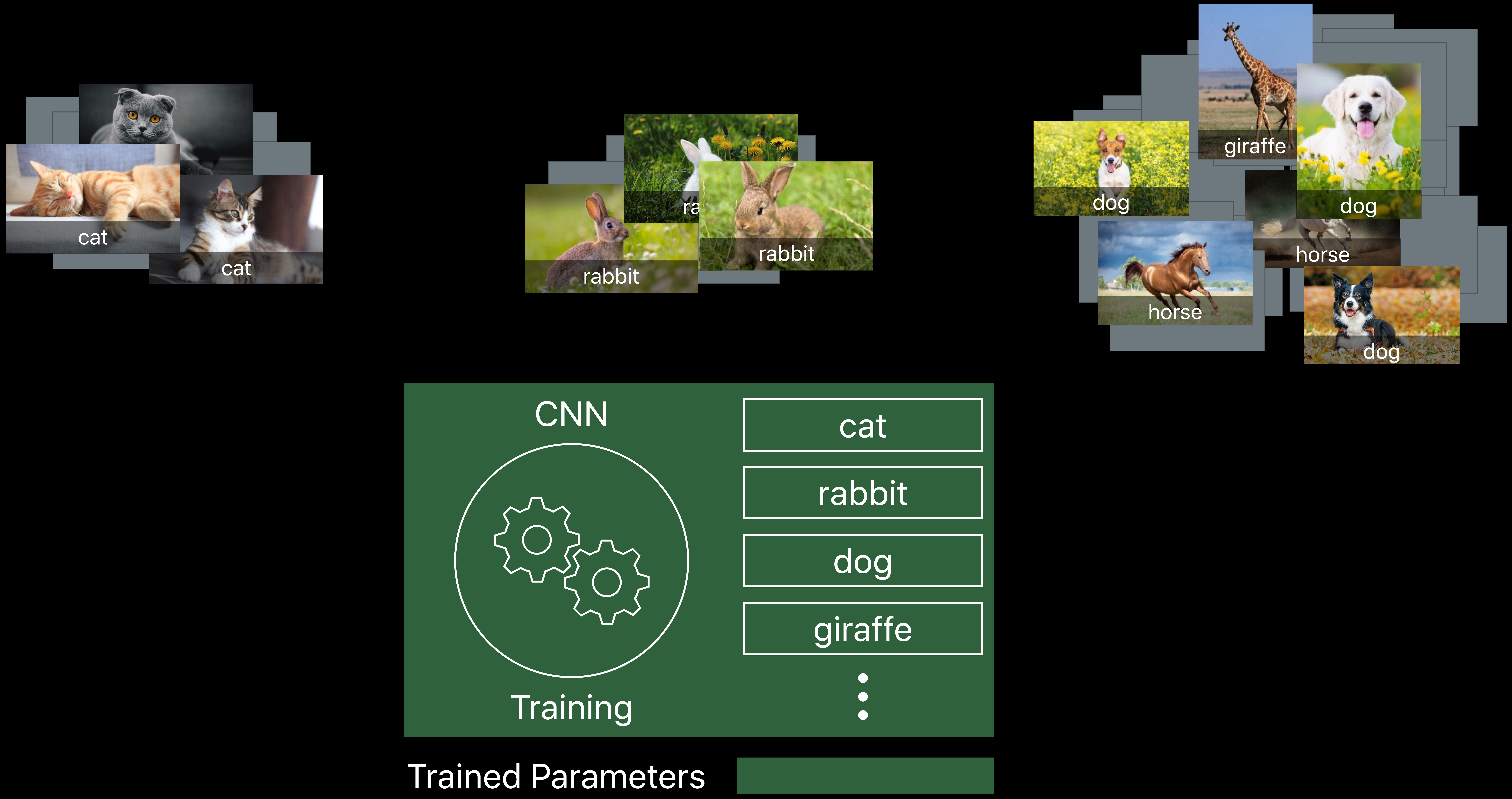
- Image processing
- Linear algebra
- Machine learning—inference and training
- Ray tracing

Training and Inference

Image classification example

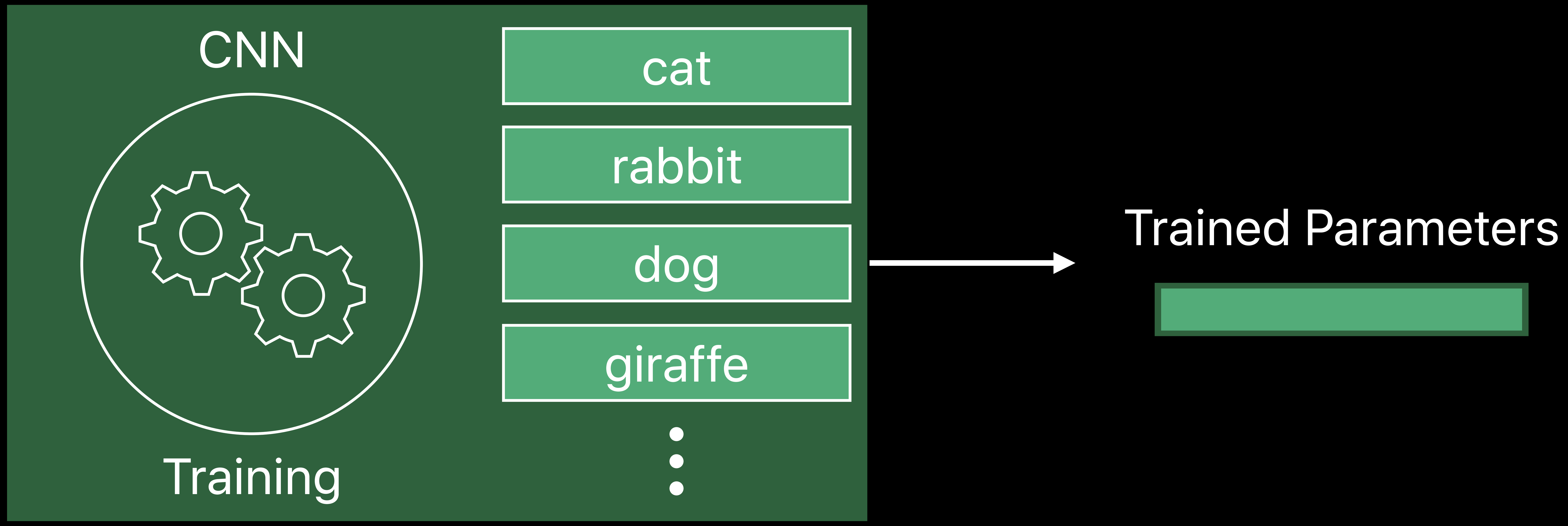
Training

Image classification example



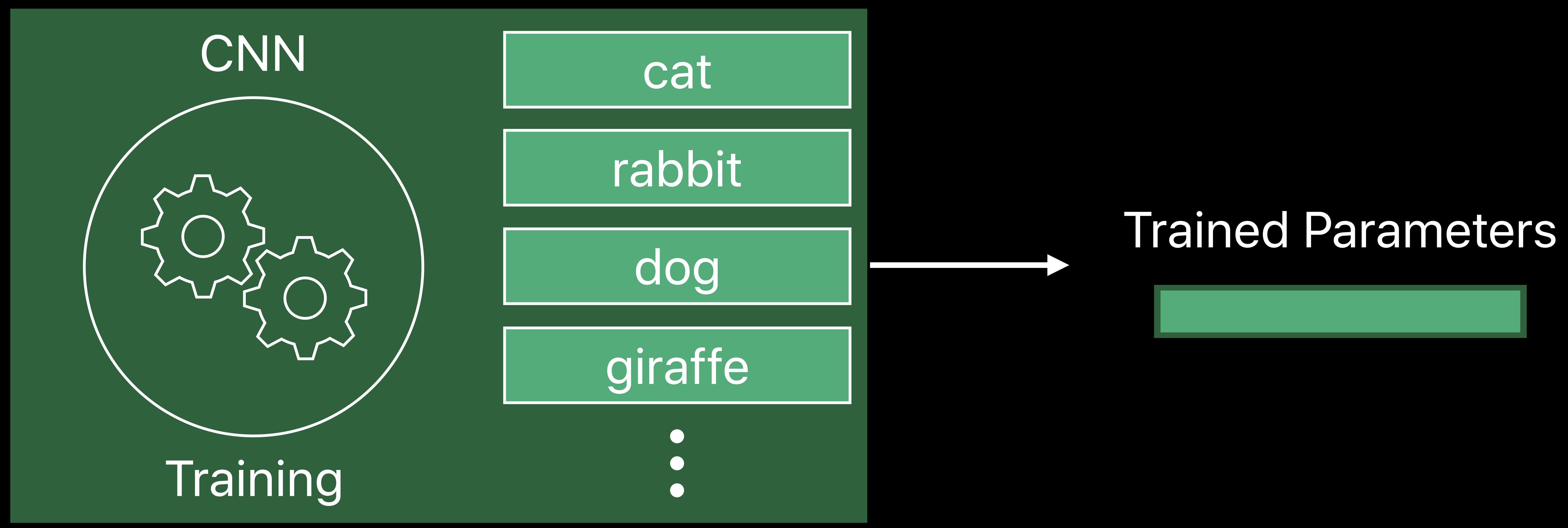
Training

Image classification example



Inference

Image classification example



Inference

Image classification example

Input Image

CNN Inference Enhancements

FP16 accumulation

NEW

Available with Apple A11 Bionic GPU for

- Convolution
- Convolution transpose

Sufficient precision for commonly used neural networks

Delivers better performance than FP32

CNN Inference Enhancements

FP16 accumulation

```
let conv = MPSCNNConvolutionNode(source: MPSNNImageNode(handle: nil),
                                 weights: MyWeights(withFile:"conv.dat"))
conv.accumulatorPrecisionOption = .half
```

CNN Inference Enhancements

FP16 accumulation

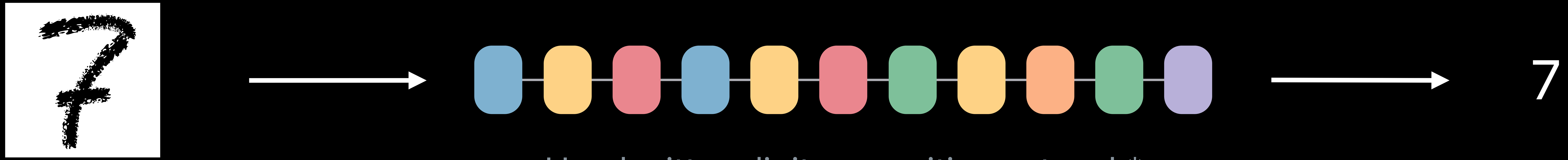
```
let conv = MPSCNNConvolutionNode(source: MPSNNImageNode(handle: nil),
                                 weights: MyWeights(withFile:"conv.dat"))
conv.accumulatorPrecisionOption = .half
```

CNN Training

Network

Handwritten digit recognition

- Convolution
- ReLu
- Max Pooling
- Fully-Connected
- Dropout
- SoftMax



Input

Handwritten digit recognition network*

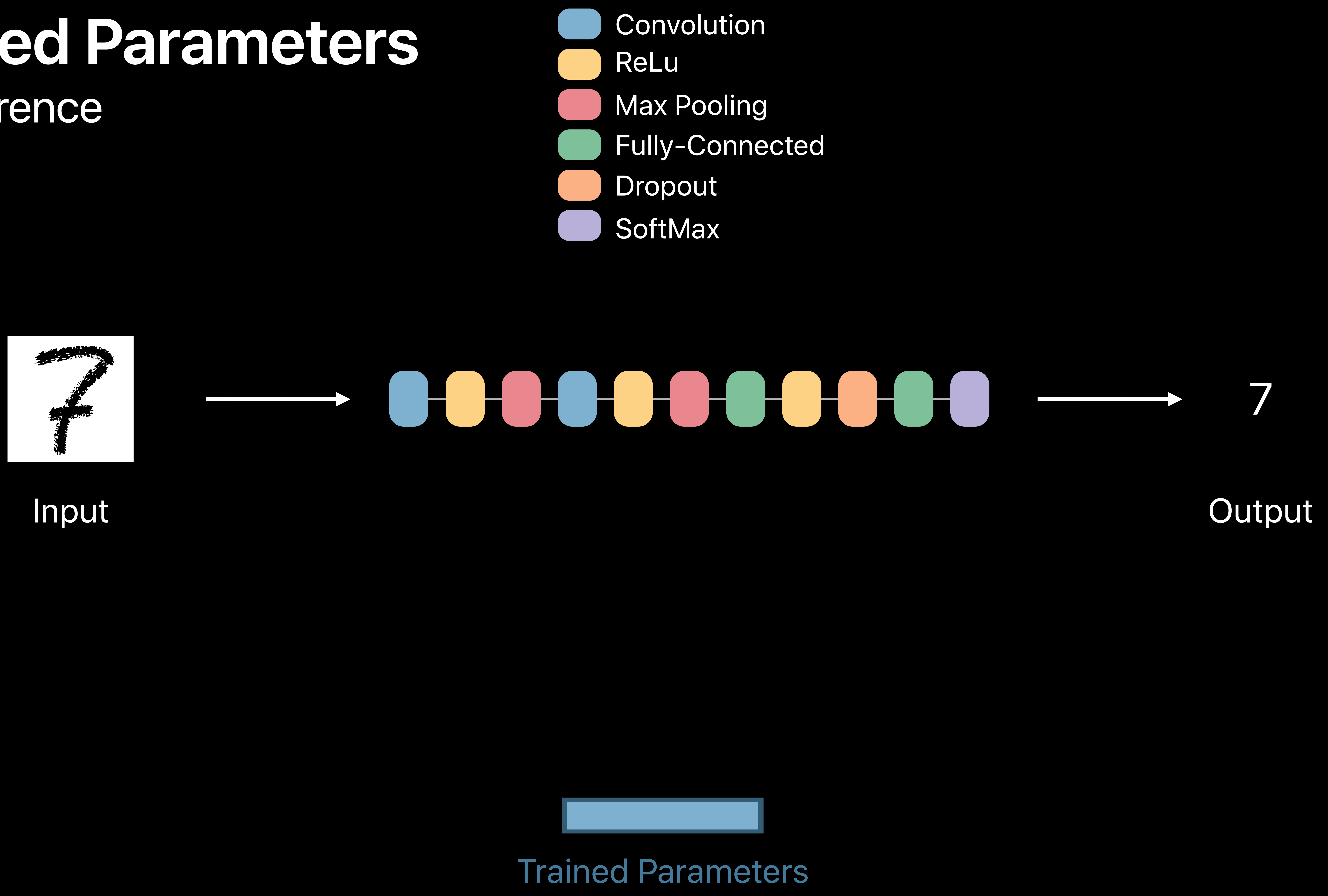
Output

Trained Parameters

*MNIST handwritten digit database, Yann LeCun, Corinna Cortes, Christopher J.C. Burges, 2013

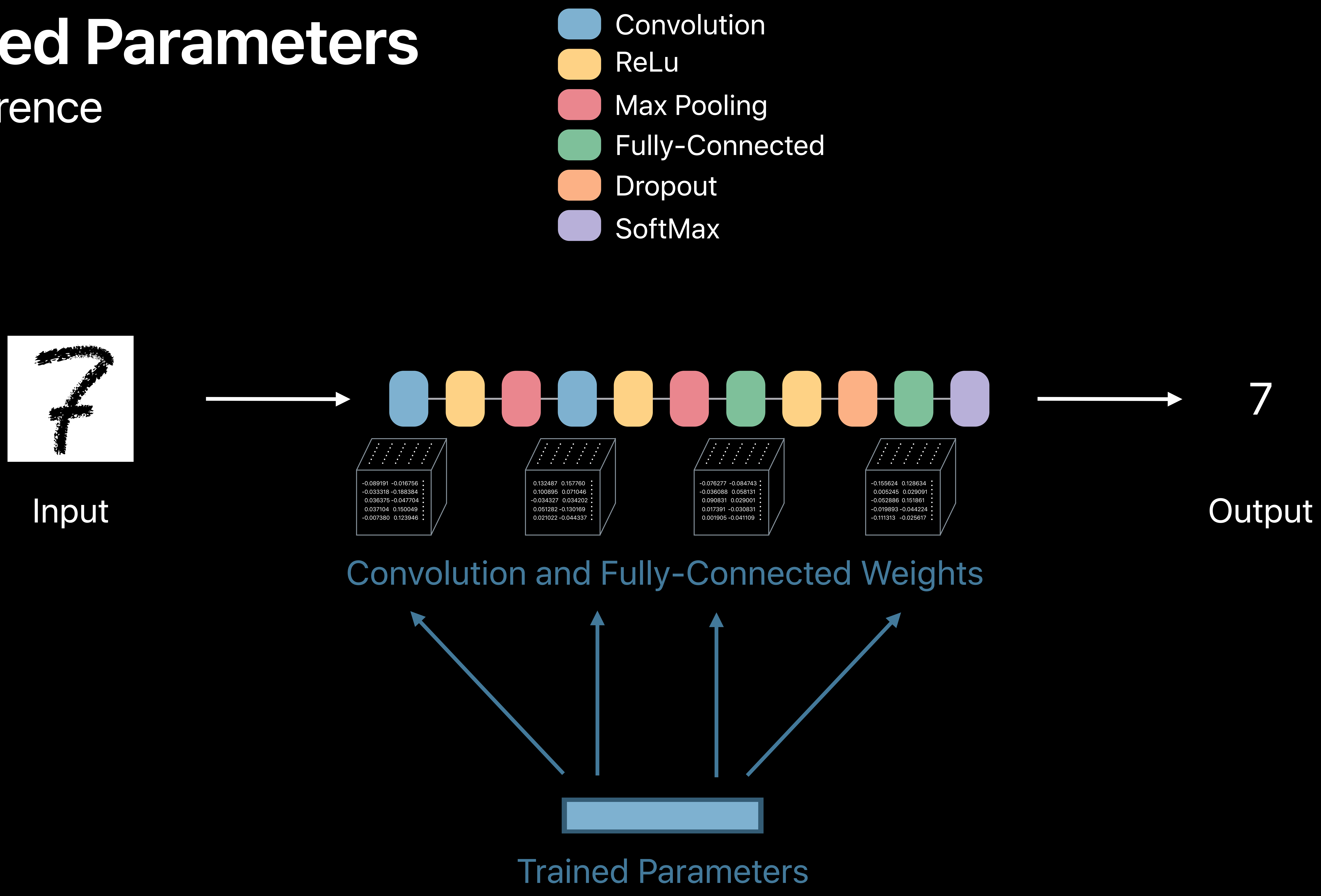
Trained Parameters

For inference



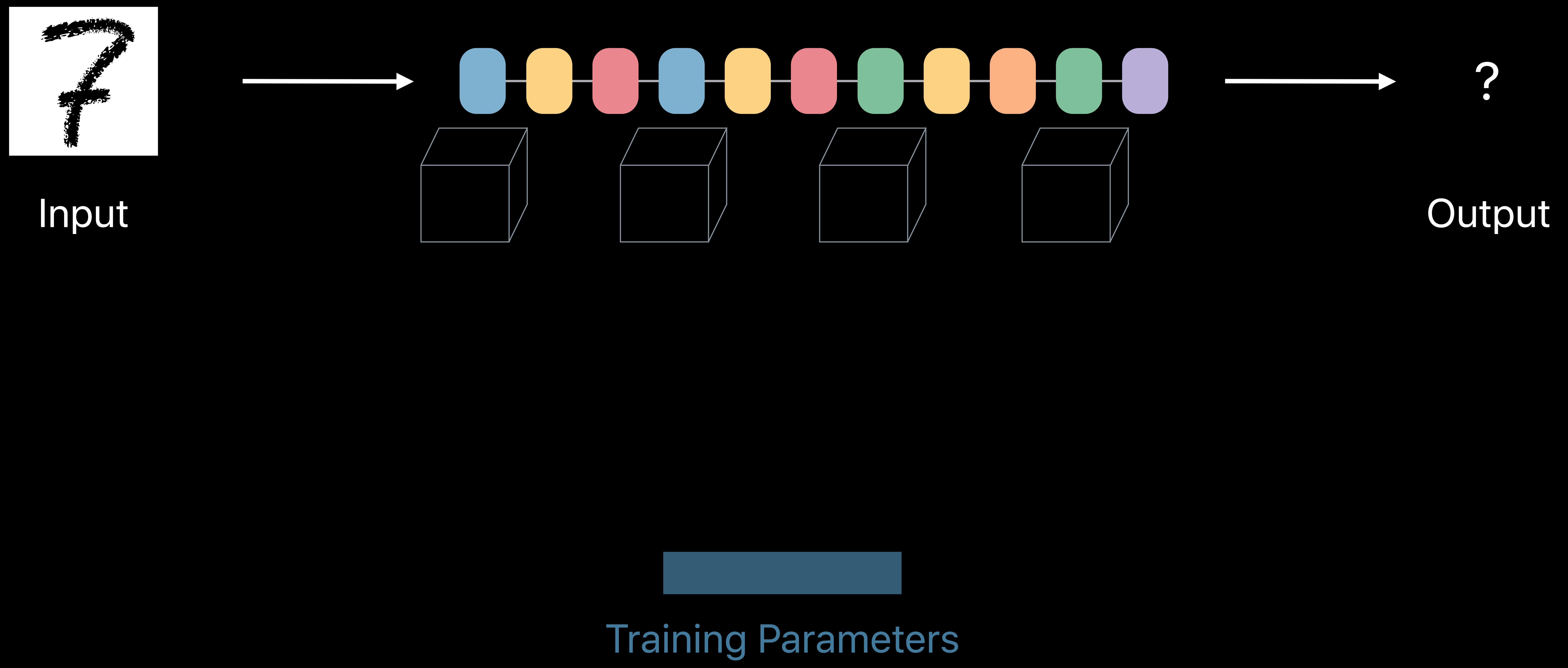
Trained Parameters

For inference



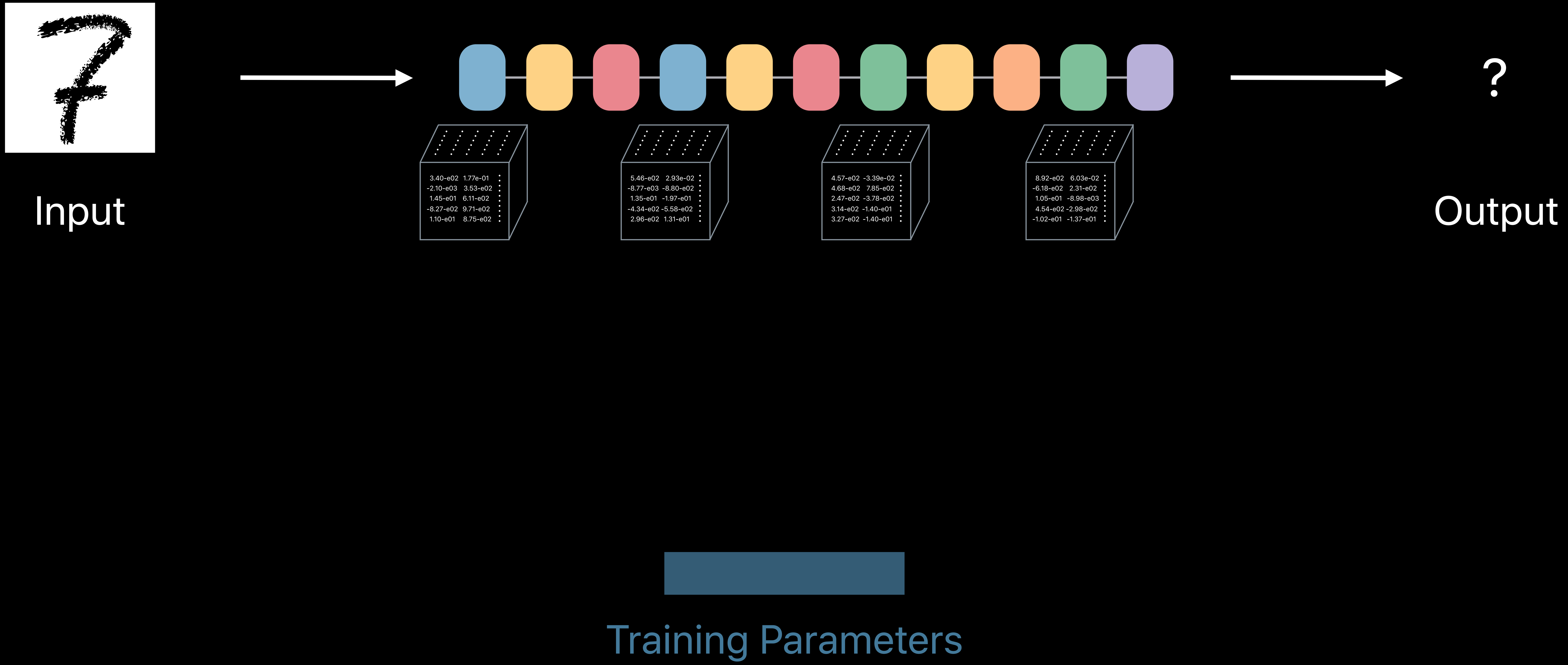
Training

Initializing weights



Training

Initializing weights

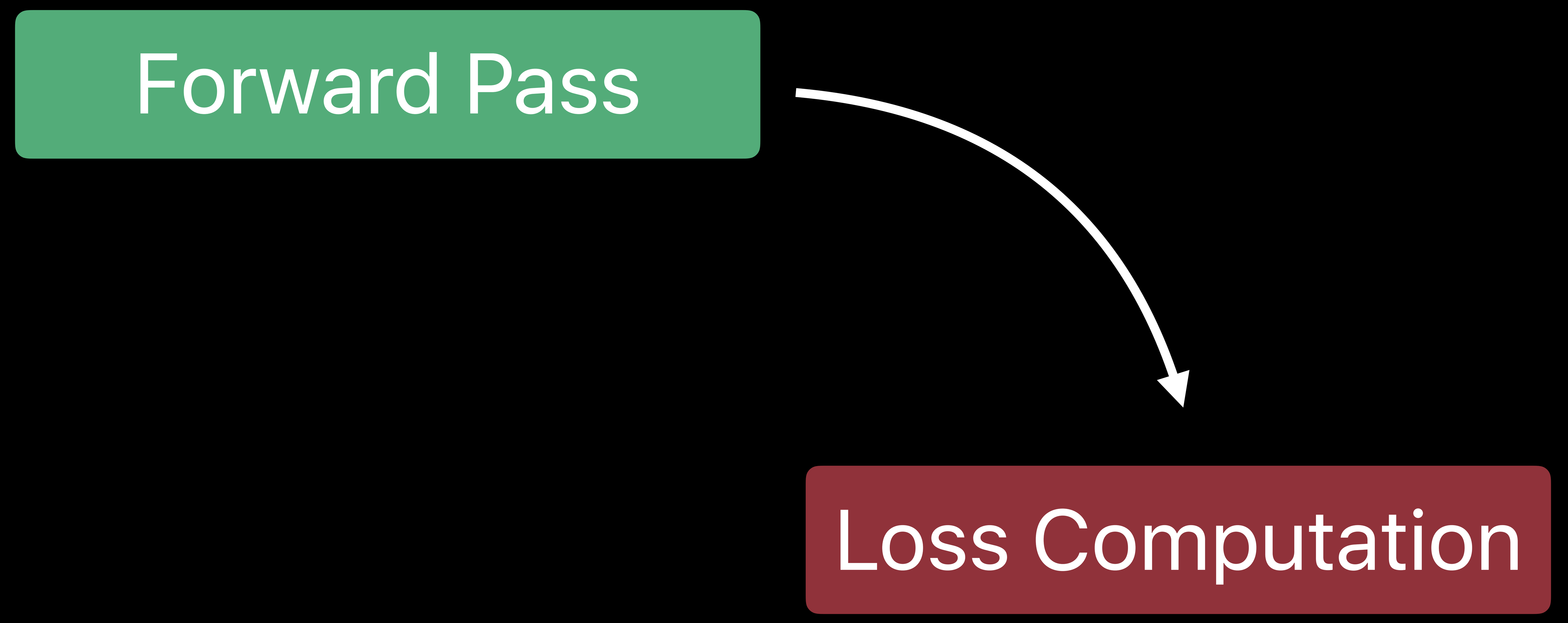


Training
Iterative process

Forward Pass

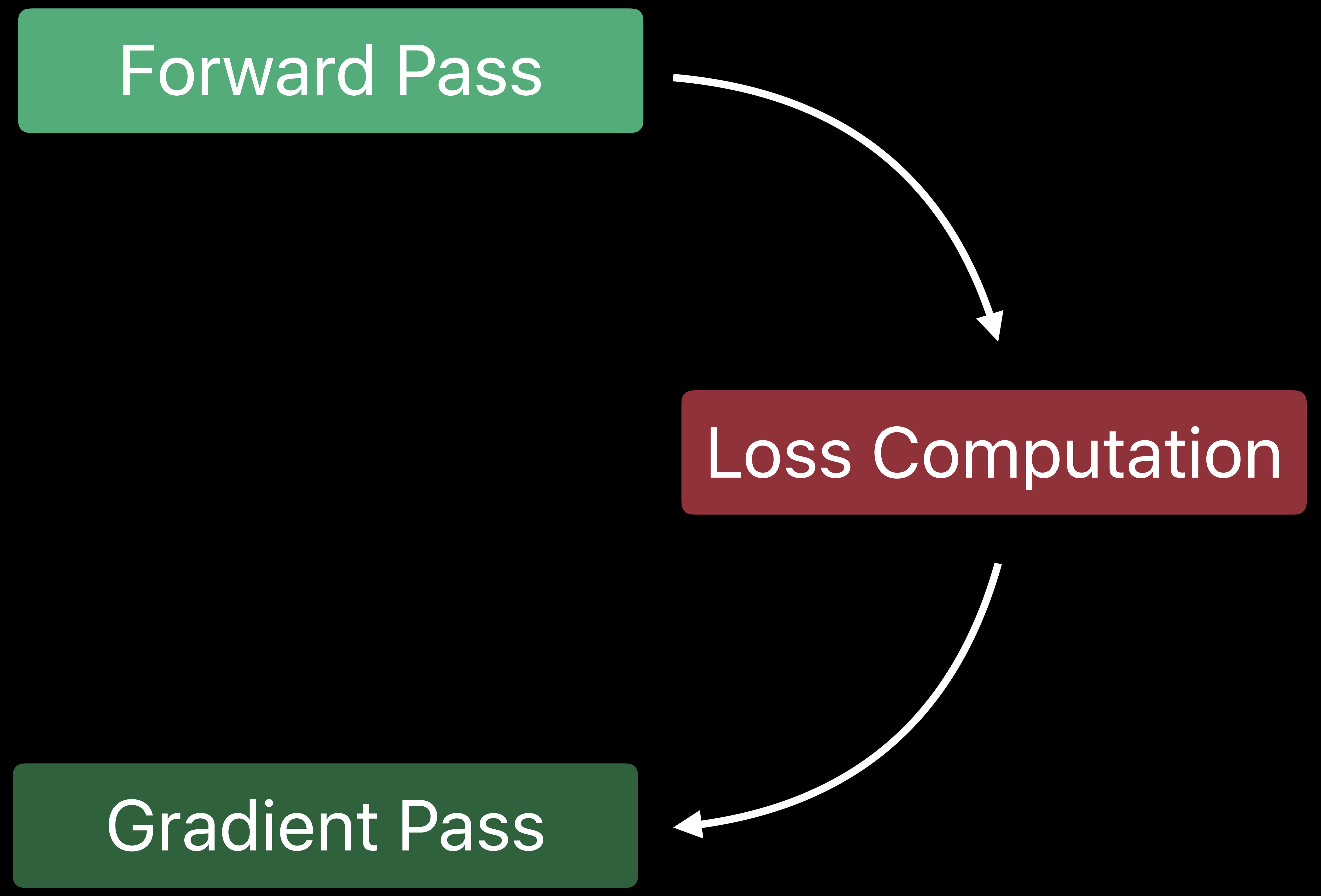
Training

Iterative process



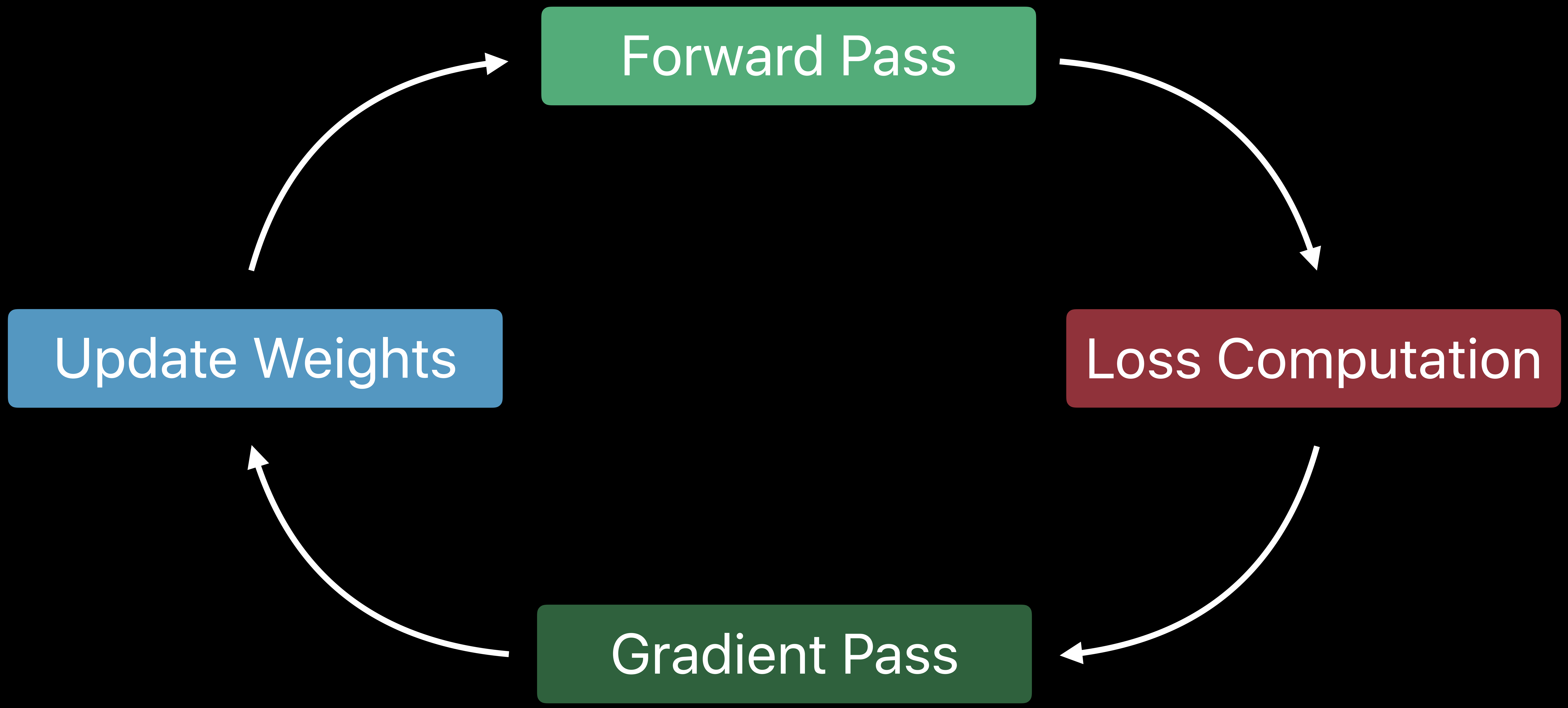
Training

Iterative process



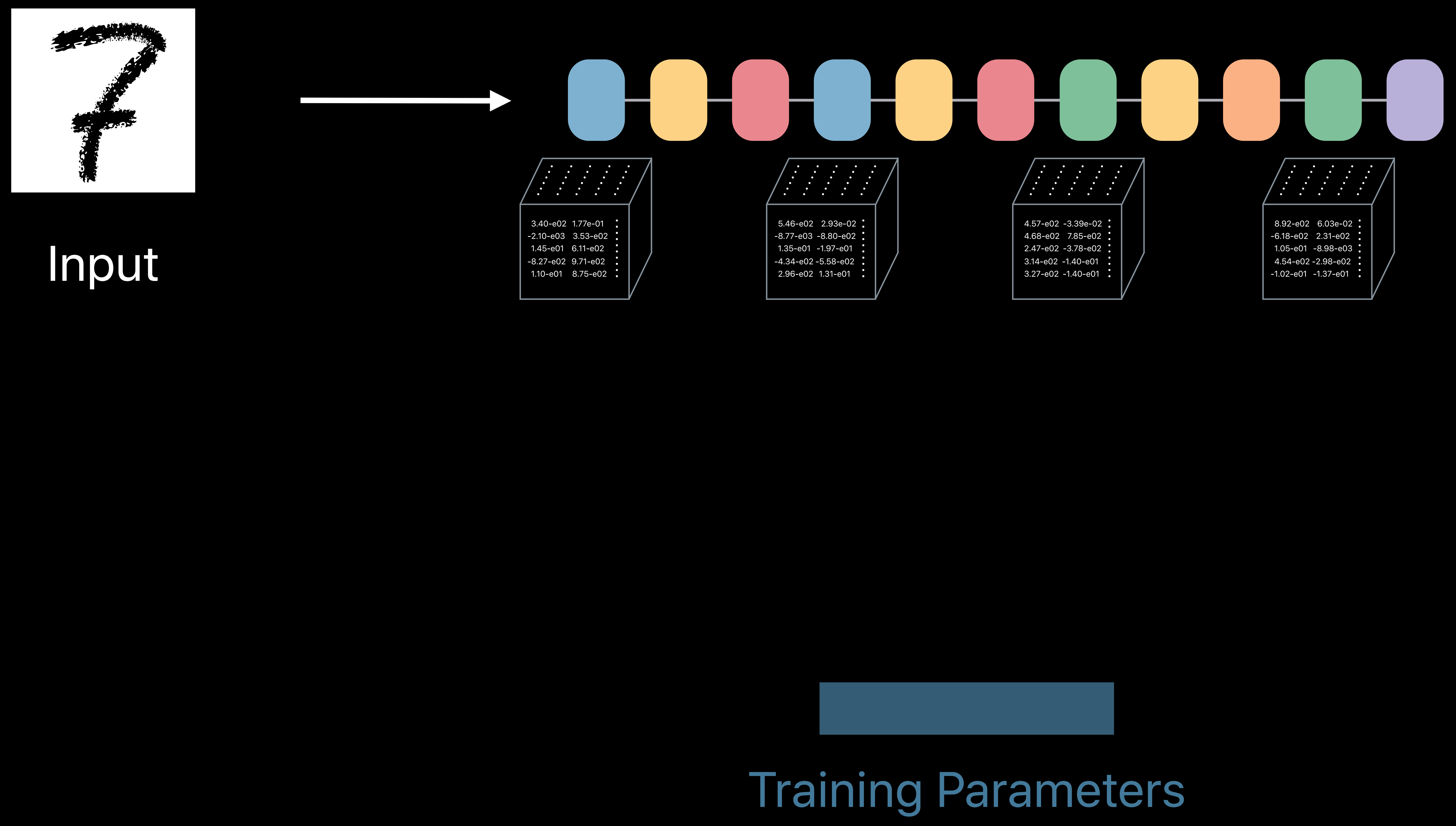
Training

Iterative process

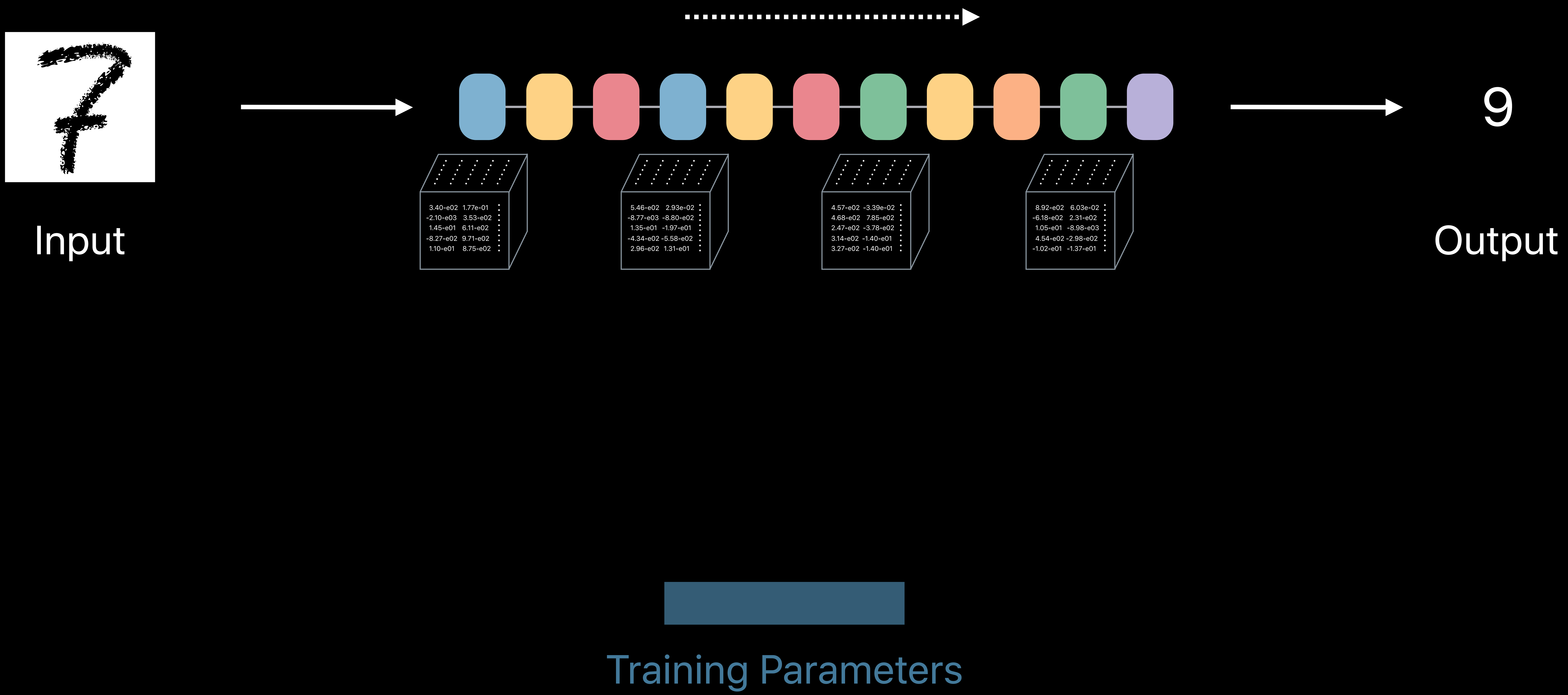


Training

Forward pass



Training
Forward pass



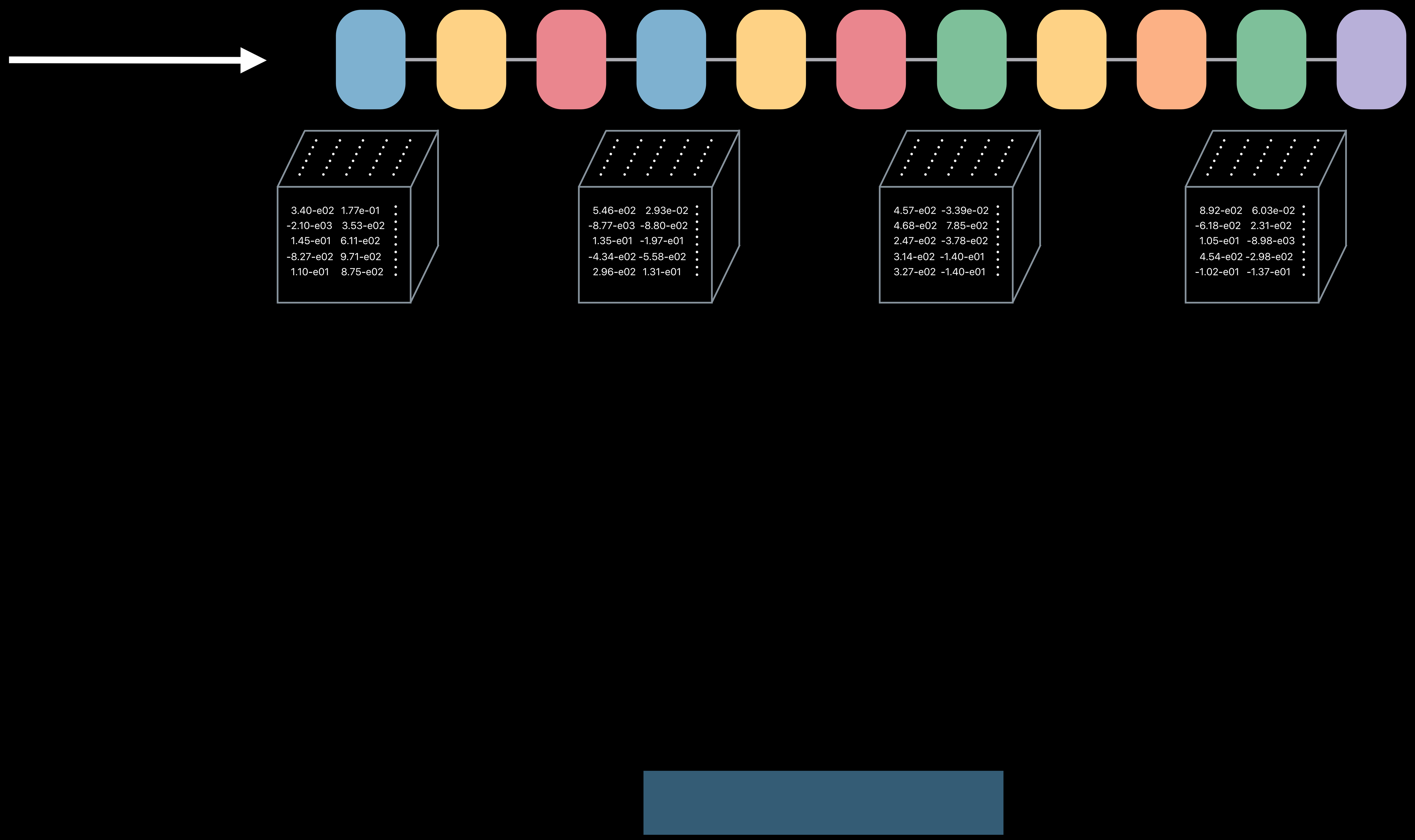
Training

Forward pass

Class	Label
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	0

Labels

Input



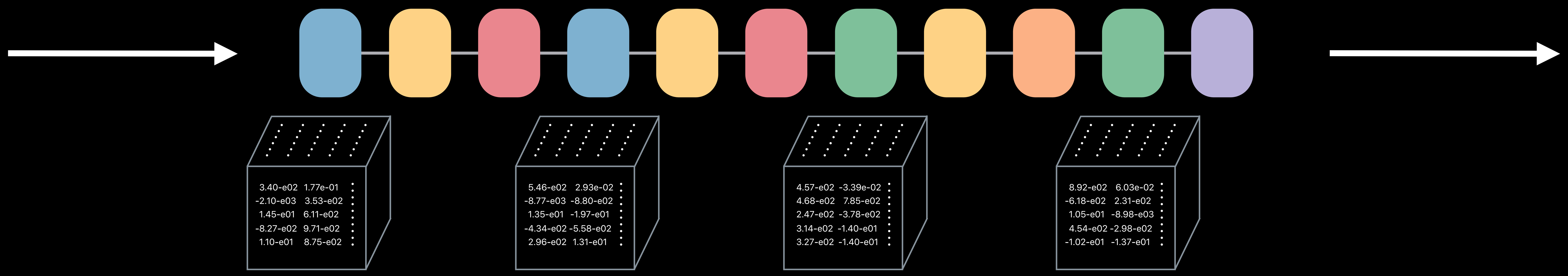
Training Parameters

Training
Forward pass

Class	Label
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	0

Labels

Input



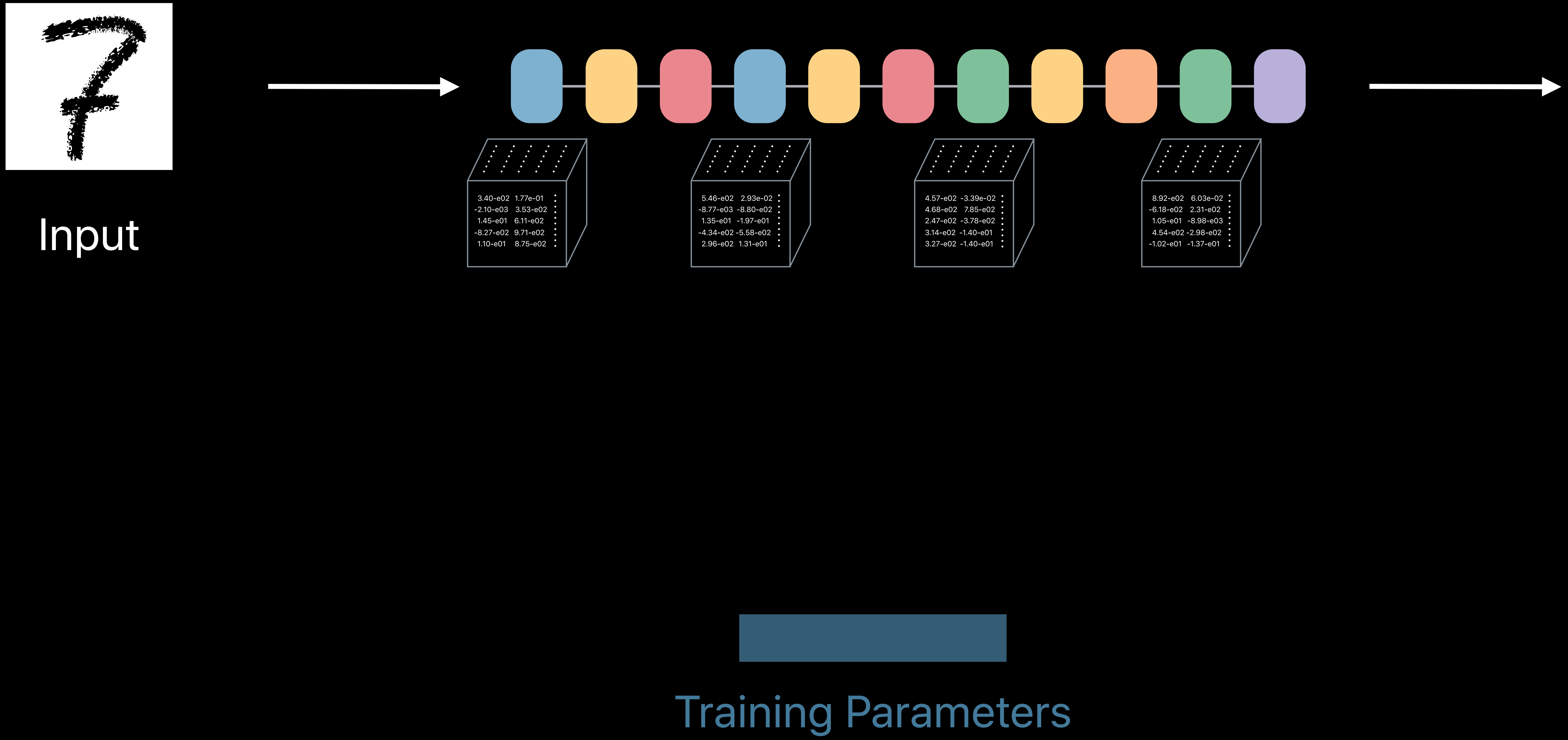
Training Parameters

Class	Probability
0	0.000045
1	0.000005
2	0.000000
3	0.015000
4	0.185300
5	0.000026
6	0.000000
7	0.000000
8	0.000126
9	0.799450

Probabilities

Training

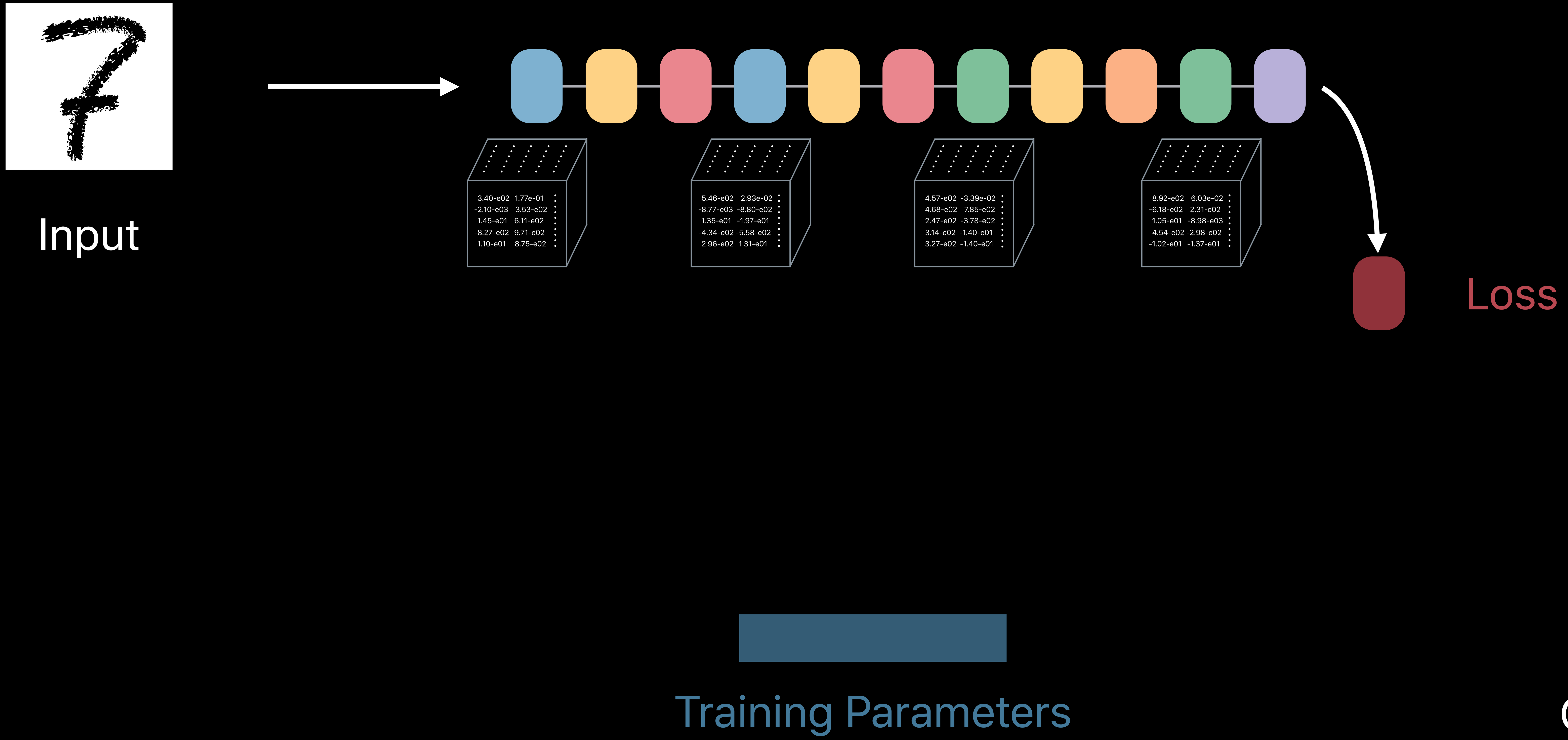
Loss computation



Class	Label	Probability
0	0	0.000045
1	0	0.000005
2	0	0.000000
3	0	0.015000
4	0	0.185300
5	0	0.000026
6	0	0.000000
7	1	0.000000
8	0	0.000126
9	0	0.799450

Training

Loss computation



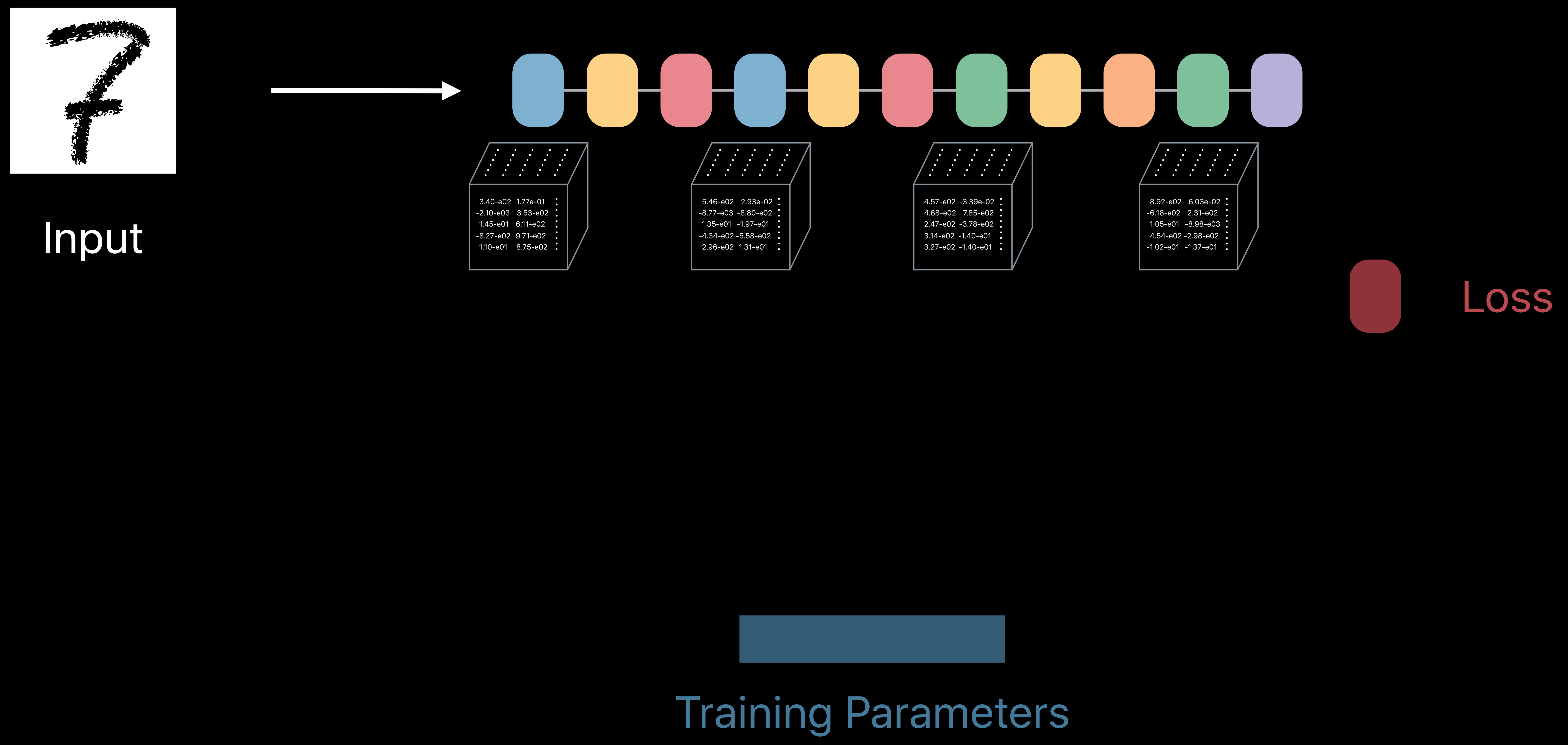
Class	Label	Probability
0	0	0.000045
1	0	0.000005
2	0	0.000000
3	0	0.015000
4	0	0.185300
5	0	0.000026
6	0	0.000000
7	1	0.000000
8	0	0.000126
9	0	0.799450

Compute loss between predictions and labels

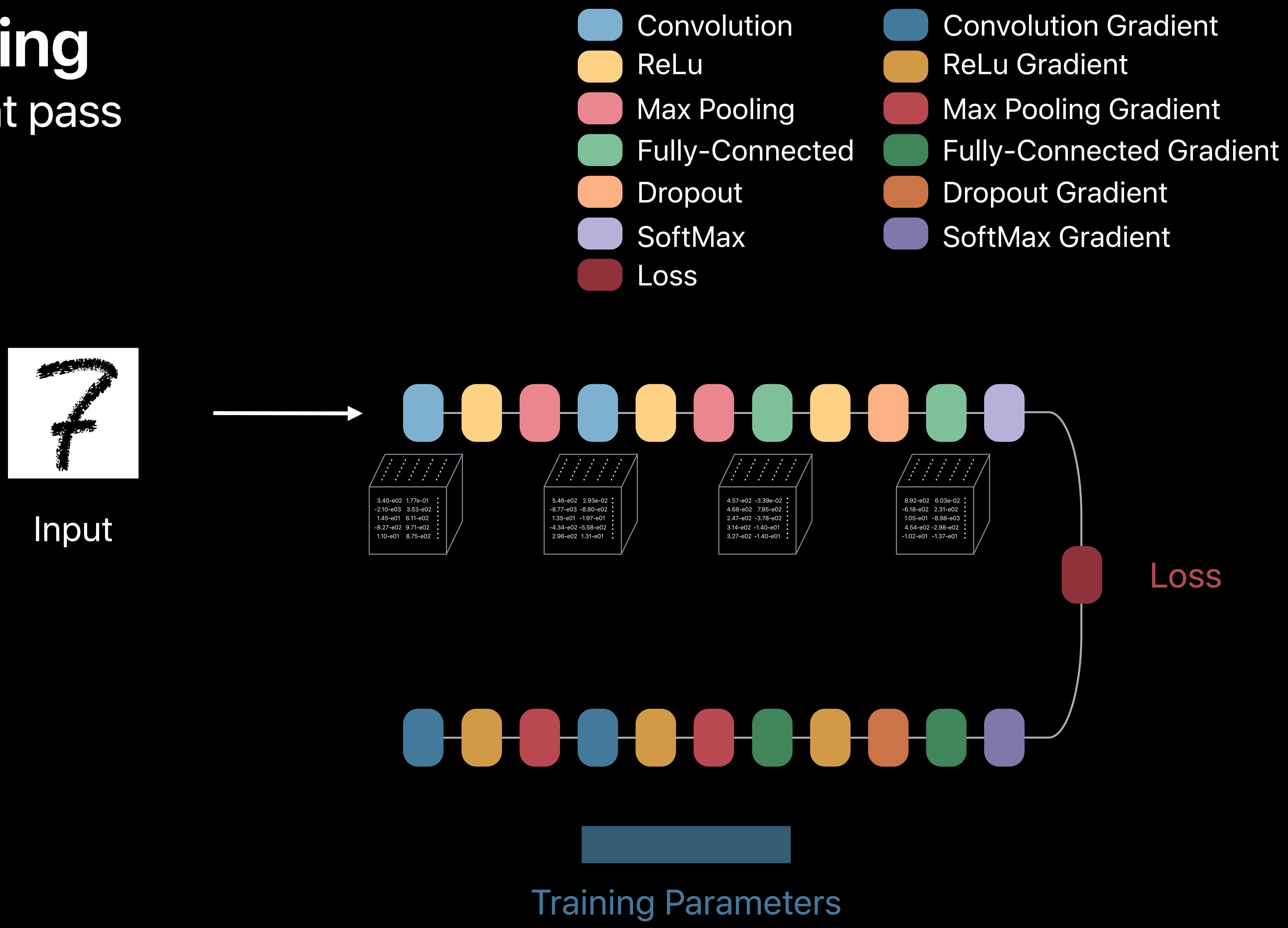
Training

Gradient pass

- Convolution
- ReLu
- Max Pooling
- Fully-Connected
- Dropout
- SoftMax
- Loss
- Convolution Gradient
- ReLu Gradient
- Max Pooling Gradient
- Fully-Connected Gradient
- Dropout Gradient
- SoftMax Gradient



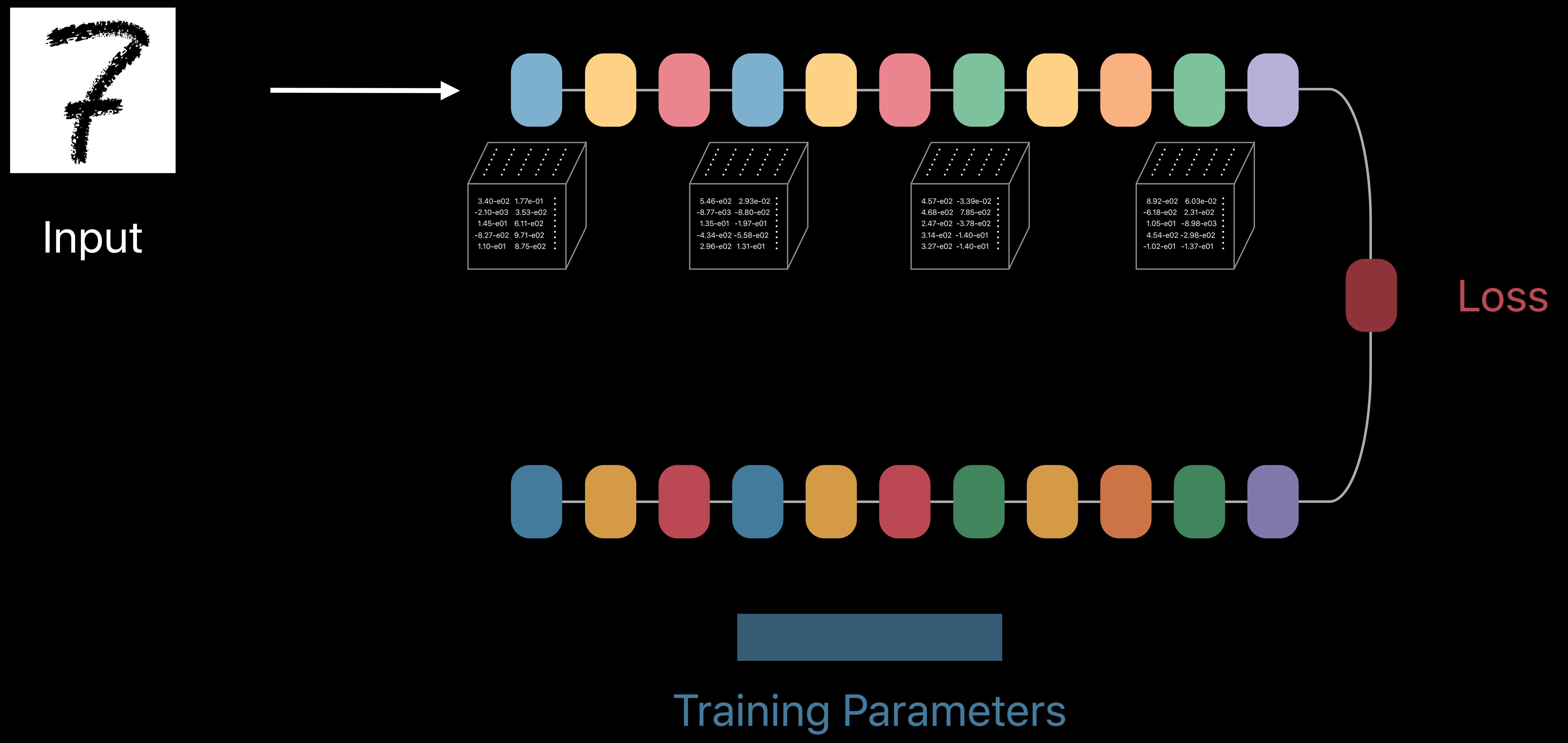
Training
Gradient pass



Training

Gradient pass

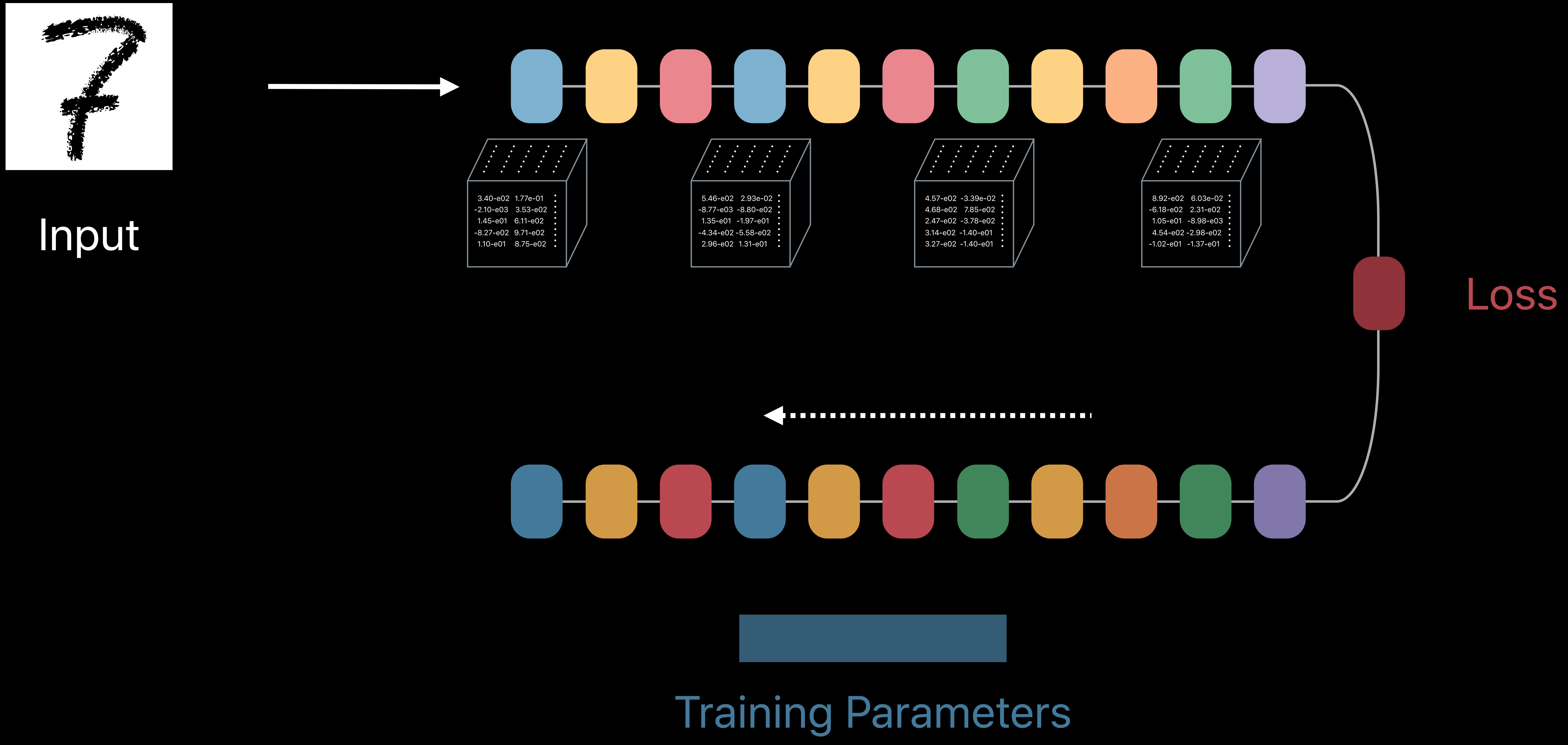
- Convolution
- ReLu
- Max Pooling
- Fully-Connected
- Dropout
- SoftMax
- Loss
- Convolution Gradient
- ReLu Gradient
- Max Pooling Gradient
- Fully-Connected Gradient
- Dropout Gradient
- SoftMax Gradient



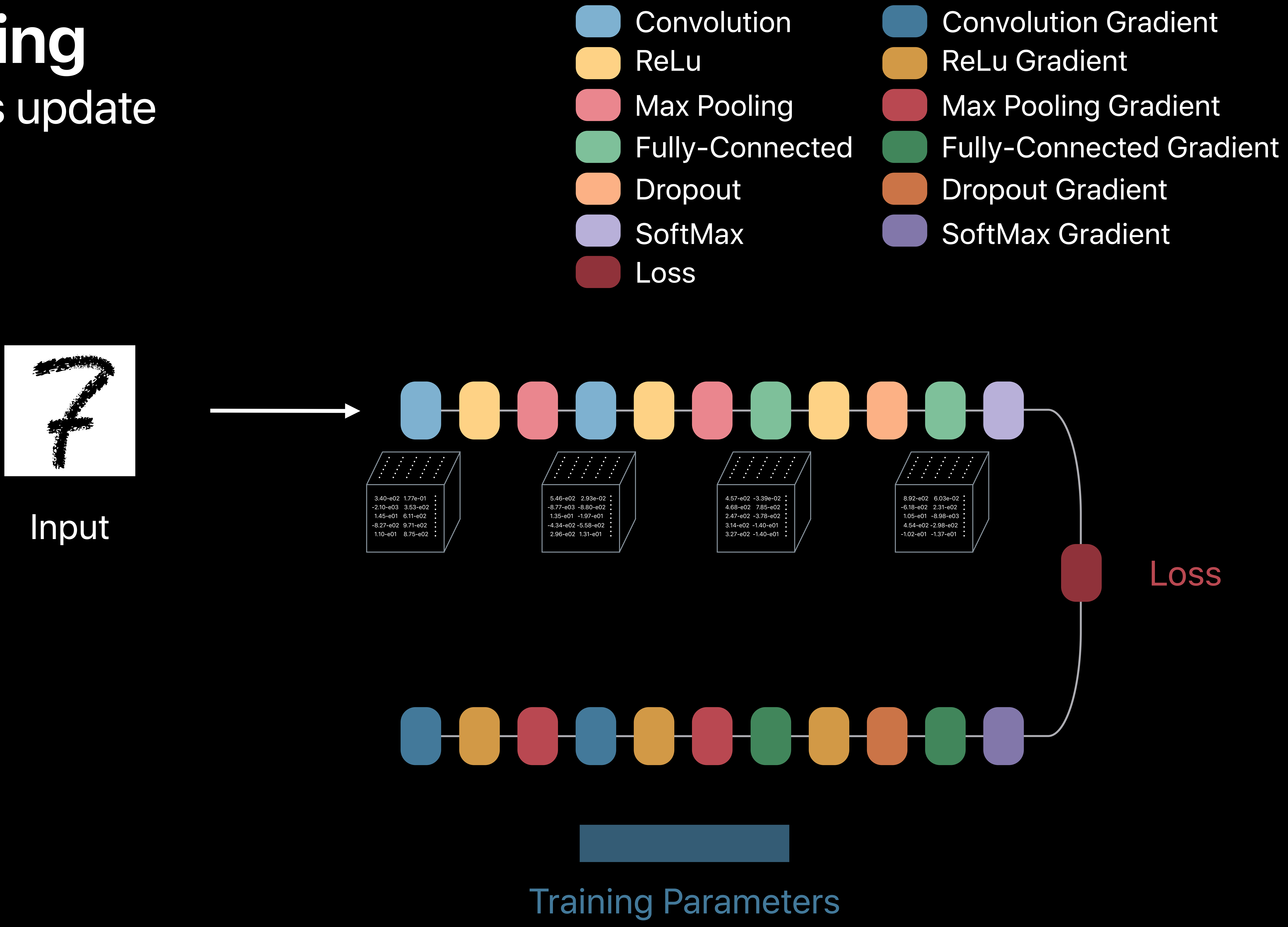
Training

Gradient pass

- Convolution
- ReLu
- Max Pooling
- Fully-Connected
- Dropout
- SoftMax
- Loss
- Convolution Gradient
- ReLu Gradient
- Max Pooling Gradient
- Fully-Connected Gradient
- Dropout Gradient
- SoftMax Gradient

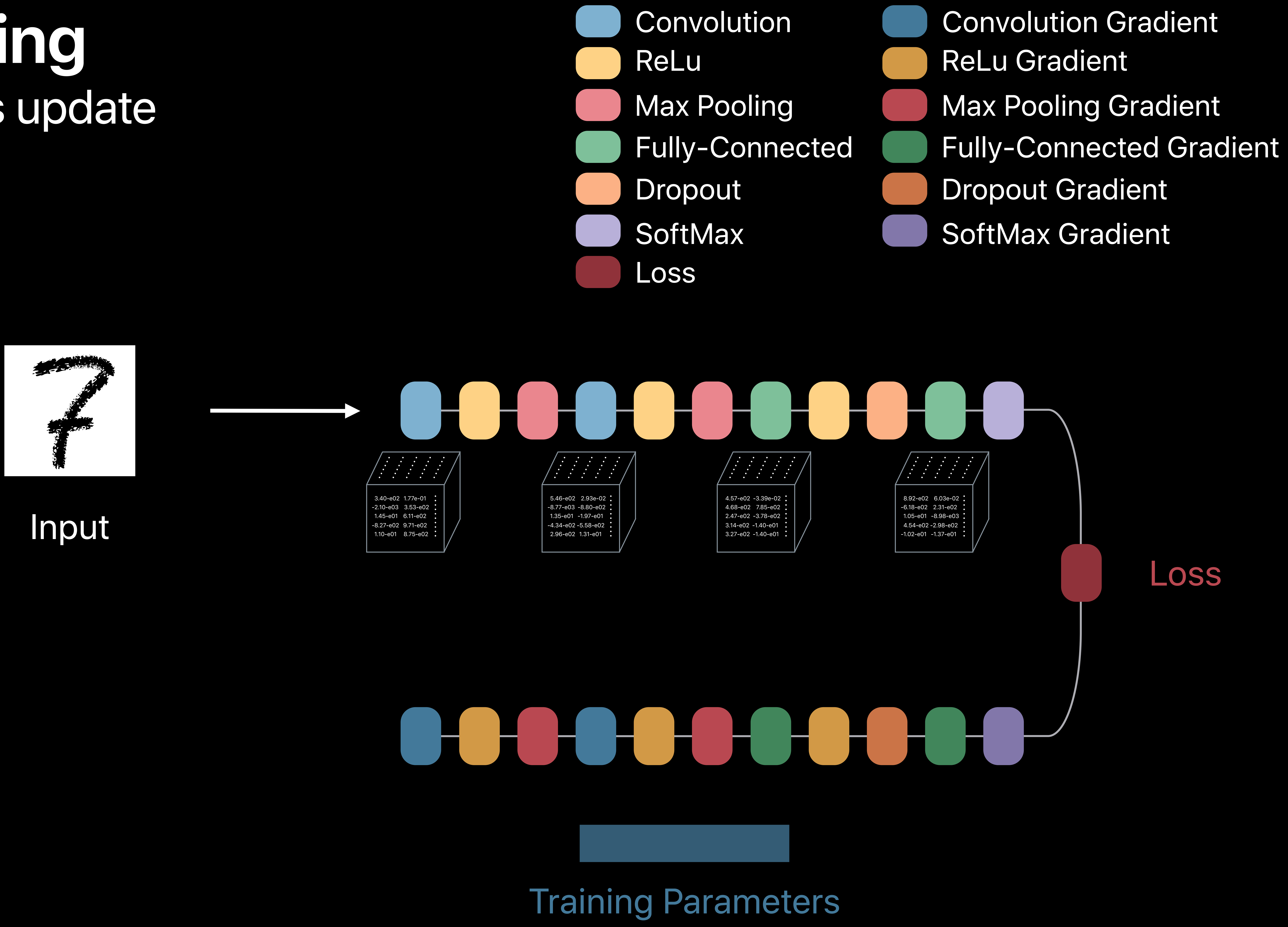


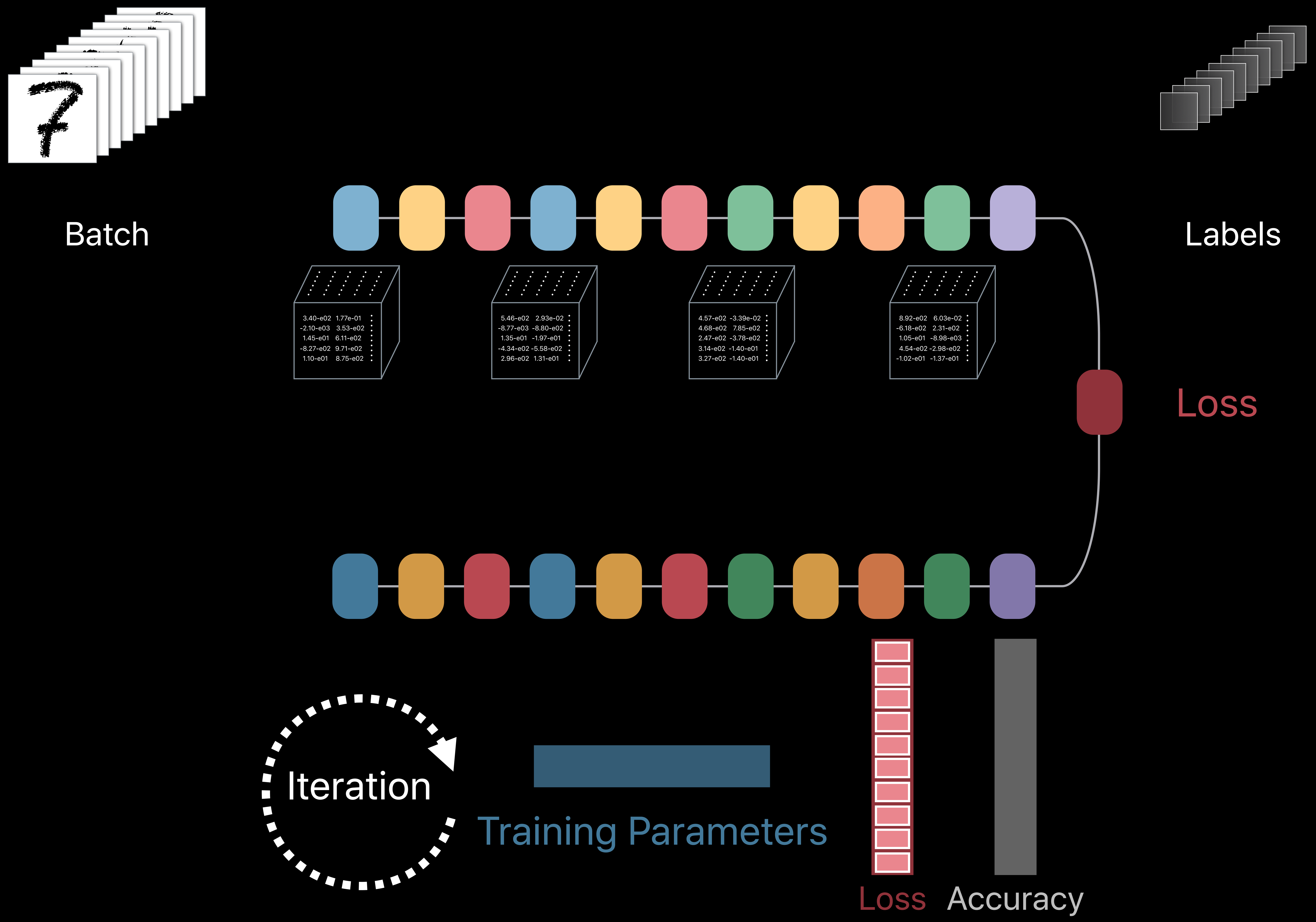
Training
Weights update



Training

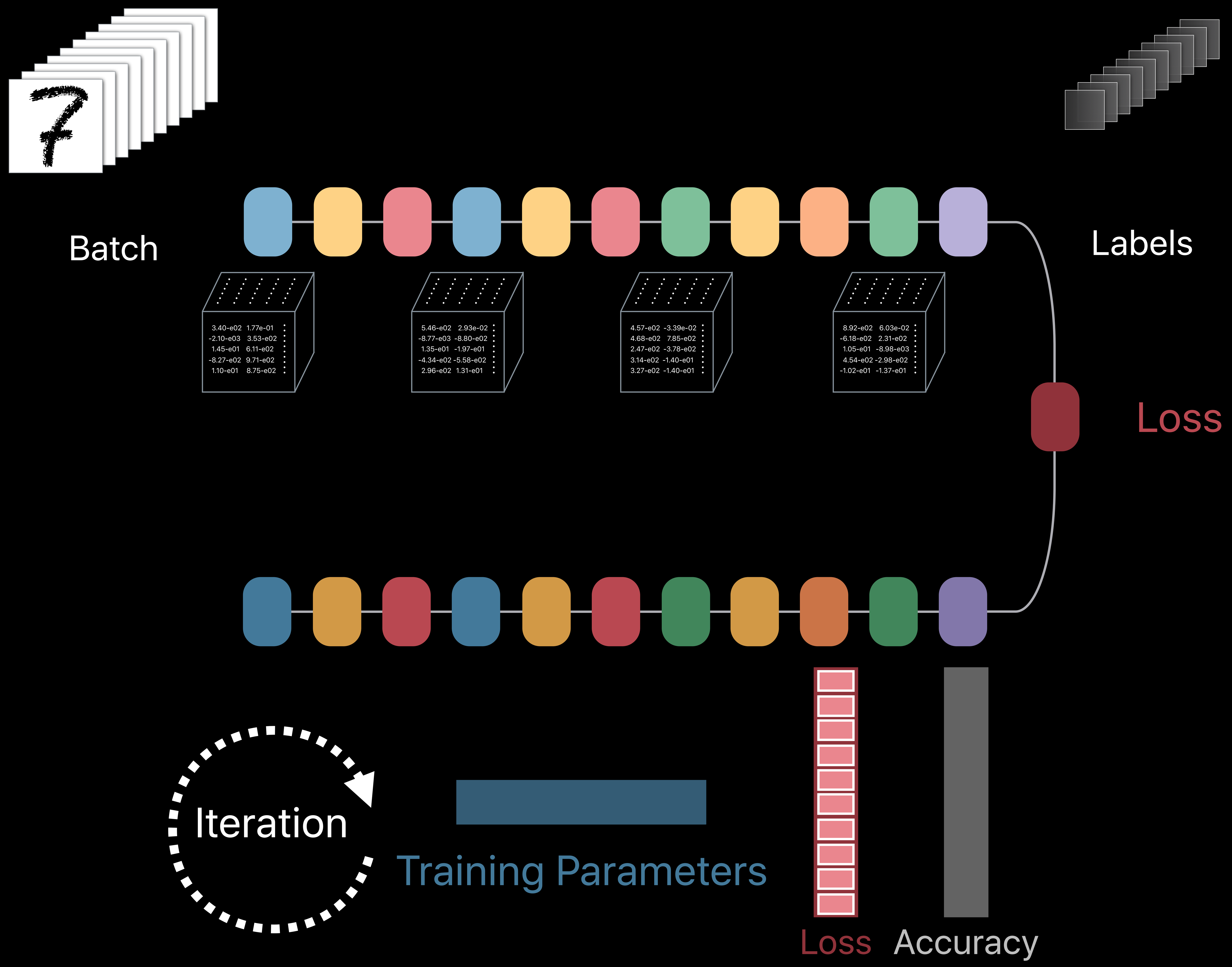
Weights update



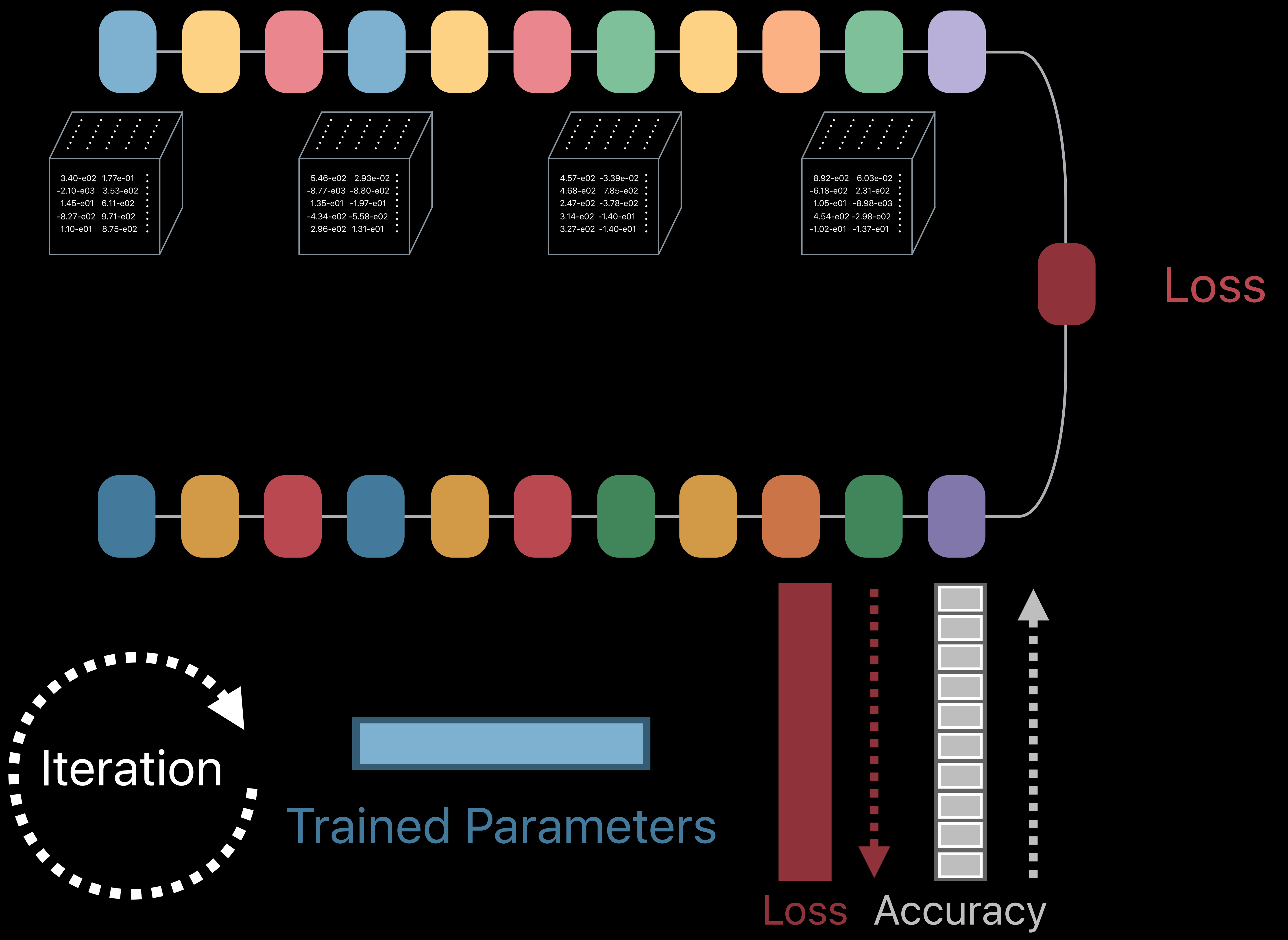


Batch
Labels
Loss
Iteration
Training Parameters
Loss
Accuracy


Batch
Labels
Loss
Iteration
Training Parameters
Loss
Accuracy


Batch
Labels
Loss
Iteration
Training Parameters
Loss
Accuracy

Batch
Batch
Batch
Batch
Labels
Labels
Labels
Labels
Loss
Iteration
Training Parameters
Loss
Accuracy


Loss
Iteration
Trained Parameters
Loss
Accuracy

Training a Neural Network with MPS

Create training graph

Prepare inputs

Specify weights

Execute graph

- Graph updates weights

Complete training process

Training a Neural Network with MPS

Create training graph

Prepare inputs

Specify weights

Execute graph

- Graph updates weights

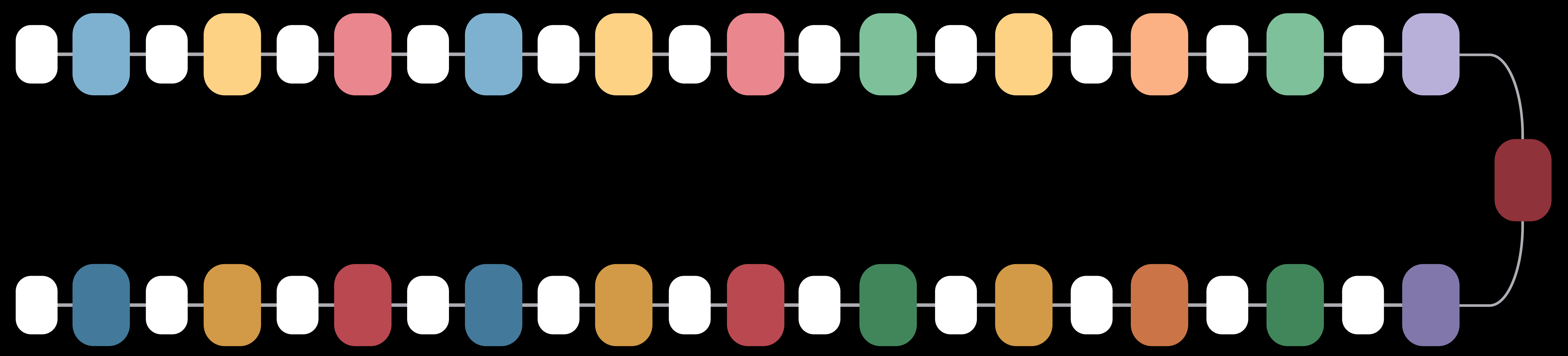
Complete training process

Create Training Graph

Neural Network Graph API

Describe neural network using graph API

- Convolution
- ReLu
- Max Pooling
- Fully Connected
- Dropout
- SoftMax
- Convolution Gradient
- ReLu Gradient
- Max Pooling Gradient
- Fully-Connected Gradient
- Dropout Gradient
- SoftMax Gradient
- Loss
- Image

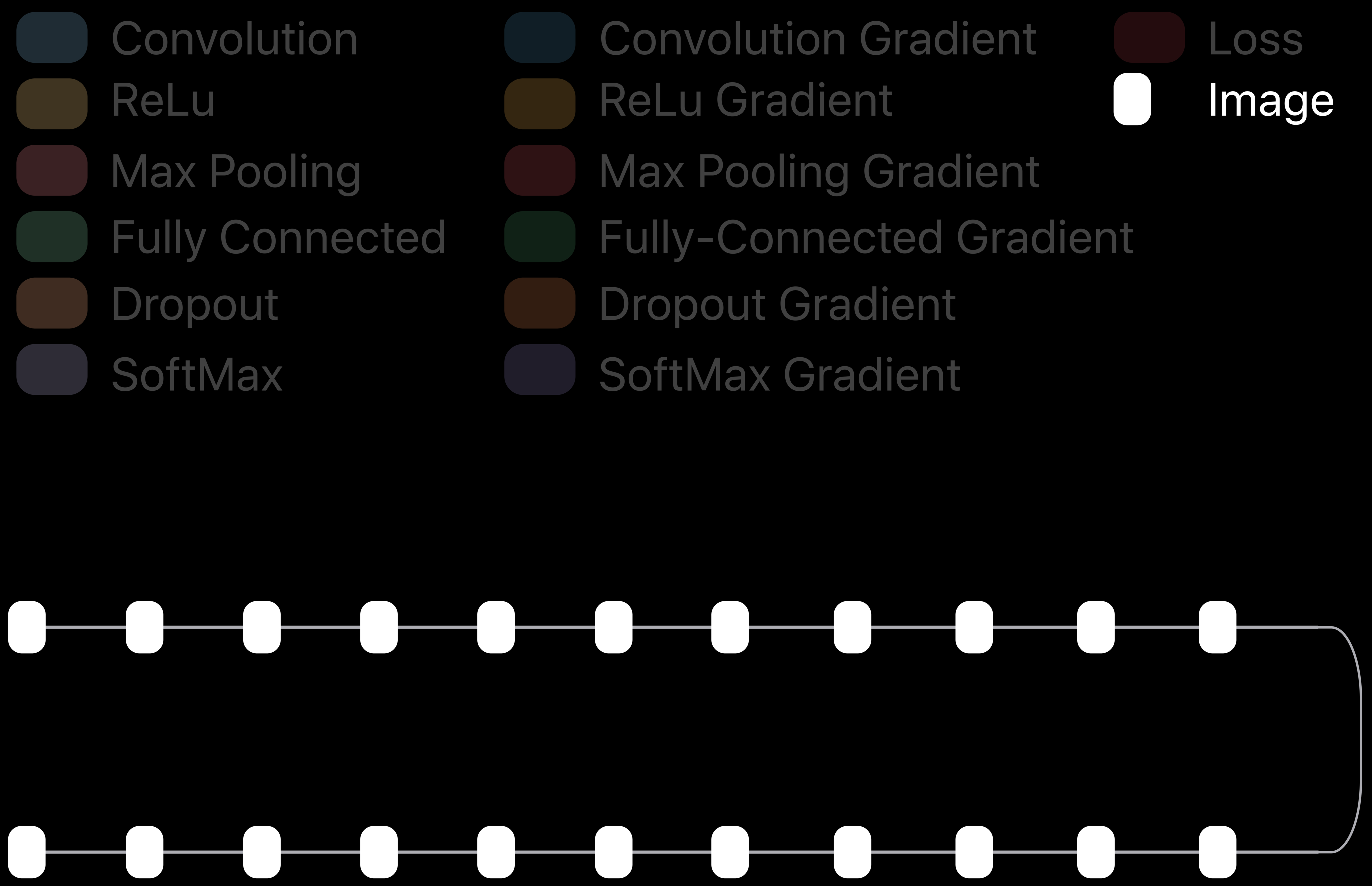


Create Training Graph

Neural Network Graph API

Describe neural network using graph API

Image nodes—Data



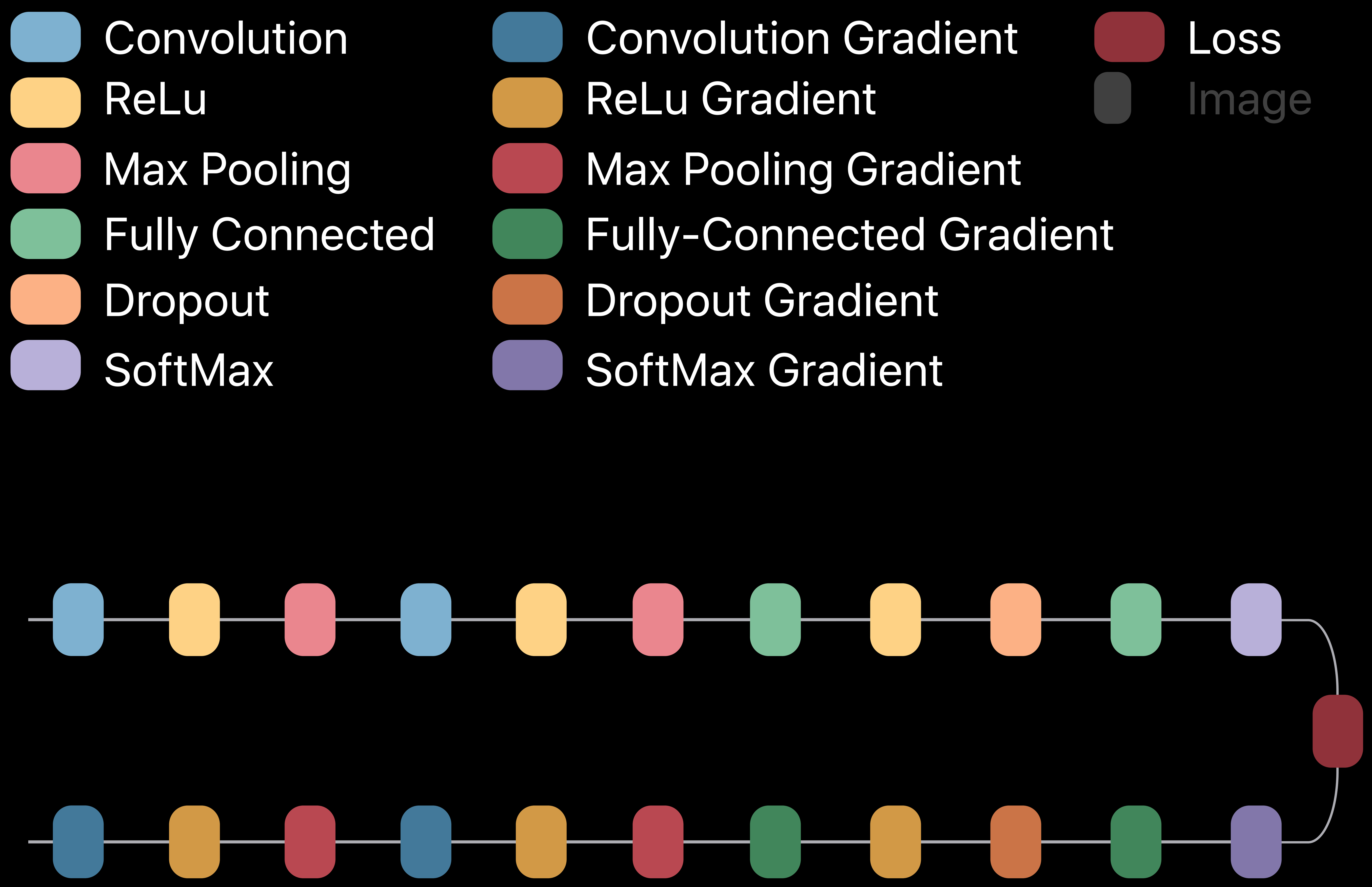
Create Training Graph

Neural Network Graph API

Describe neural network using graph API

Image nodes—Data

Filter nodes—Operations



Create Training Graph

Neural Network Graph API

Describe neural network using graph API

Image nodes—Data

Filter nodes—Operations

```
let inputImage = MPSNNImageNode(handle: nil)
let conv = MPSCNNConvolutionNode(source: inputImage,
                                 weights: MyWeights(withFile:"conv.dat"))
let convGradient = conv.gradientFilter(source: previousNode.resultImage)
```

Create Training Graph

Neural Network Graph API

Describe neural network using graph API

Image nodes—Data

Filter nodes—Operations

```
let inputImage = MPSNNImageNode(handle: nil)
let conv = MPSCNNConvolutionNode(source: inputImage,
                                 weights: MyWeights(withFile:"conv.dat"))
let convGradient = conv.gradientFilter(source: previousNode.resultImage)
```

Create Training Graph

Neural Network Graph API

Describe neural network using graph API

Image nodes—Data

Filter nodes—Operations

```
let inputImage = MPSNNImageNode(handle: nil)
let conv = MPSCNNConvolutionNode(source: inputImage,
                                 weights: MyWeights(withFile:"conv.dat"))
let convGradient = conv.gradientFilter(source: previousNode.resultImage)
```

Create Training Graph

Neural Network Graph API

NEW

Describe neural network using graph API

Image nodes—Data

Filter nodes—Operations

```
let inputImage = MPSNNImageNode(handle: nil)
let conv = MPSCNNConvolutionNode(source: inputImage,
                                 weights: MyWeights(withFile:"conv.dat"))
let convGradient = conv.gradientFilter(source: previousNode.resultImage)
```

```swift
// Example: Create an Inference Graph
func makeInferenceGraph() -> 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 fc1 = MPSCNNFullyConnectedNode(source: pool2.resultImage, weights: MyWeights(file:"fc1.dat"))
    let fc2 = MPSCNNFullyConnectedNode(source:   fc1.resultImage, weights: MyWeights(file:"fc2.dat"))
    return                                       fc2.resultImage
}
```

```swift
// Example: Create an Inference Graph
func makeInferenceGraph() -> 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 fc1 PSCNNFullyConnectedNode(source: pool2.resultImage, weights: MyWeights(file:"fc1.dat"))
    let fc2 PSCNNFullyConnectedNode(source:   fc1.resultImage, weights: MyWeights(file:"fc2.dat"))
    return                                    fc2.resultImage
}
```

```swift
// Example: Create an Inference Graph
func makeInferenceGraph() -> 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 fc1 = MPSCNNFullyConnectedNode(sourc pool2.resultImage, weights: MyWeights(file:"fc1.dat"))
    let fc2 = MPSCNNFullyConnectedNode(sourc fc1.resultImage, weights: MyWeights(file:"fc2.dat"))
    return fc2.resultImage
}
```

```swift
// Example: Create a Training Graph
func makeTrainingGraph() -> 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 fc1 = MPSCNNFullyConnectedNode(source: pool2.resultImage, weights: MyWeights(file:"fc1.dat"))
    let fc2 = MPSCNNFullyConnectedNode(source: fc1.resultImage, weights: MyWeights(file:"fc2.dat"))
    let loss  = MPSCNNLossNode(source: fc2.resultImage, lossDescriptor: lossDescriptor)
    . . .
}
```

```swift
// Example: Create a Training Graph
func makeTrainingGraph() -> 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 fc1 = MPSCNNFullyConnectedNode(sourc pool2.resultImage, weights: MyWeights(file:"fc1.dat"))
    let fc2 = MPSCNNFullyConnectedNode(sourc fc1.resultImage, weights: MyWeights(file:"fc2.dat"))
    let loss MPSCNNLossNode(source: fc2.resultImage, lossDescriptor: lossDescriptor)
    . . .
}
```

```swift
// Example: Create a Training Graph
func makeTrainingGraph() -> 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 fc1 = MPSCNNFullyConnectedNode(sourc pool2.resultImage, weights: MyWeights(file:"fc1.dat"))
    let fc2 = MPSCNNFullyConnectedNode(sourc fc1.resultImage, weights: MyWeights(file:"fc2.dat"))
    let loss MPSCNNLossNode(source: fc2.resultImage, lossDescriptor: lossDescriptor)
    let fc2G  =    fc2.gradientFilter(source:   loss.resultImage)
    let fc1G =     fc1.gradientFilter(source:   fc2G.resultImage)
    let pool2G = pool2.gradientFilter(source:   fc1G.resultImage)
    let conv2G = conv2.gradientFilter(source: pool2G.resultImage)
    let pool1G = pool1.gradientFilter(source: conv2G.resultImage)
    let conv1G = conv1.gradientFilter(source: pool1G.resultImage)
   return                                   conv1G.resultImage
}
```

```swift
// Example: Create a Training Graph
func makeTrainingGraph() -> 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 fc1 = MPSCNNFullyConnectedNode(source: pool2.resultImage, weights: MyWeights(file:"fc1.dat"))
    let fc2 = MPSCNNFullyConnectedNode(source: fc1.resultImage, weights: MyWeights(file:"fc2.dat"))
    let loss  = MPSCNNLossNode(source: fc2.resultImage, lossDescriptor: lossDescriptor)
    let fc2G  =    fc2.gradientFilter(source: loss.resultImage)
    let fc1G =     fc1.gradientFilter(source: fc2G.resultImage)
    let pool2G = pool2.gradientFilter(source: fc1G.resultImage)
    let conv2G = conv2.gradientFilter(source: pool2G.resultImage)
    let pool1G = pool1.gradientFilter(source: conv2G.resultImage)
    let conv1G = conv1.gradientFilter(source: pool1G.resultImage)
  return conv1G.resultImage
}
```

Graph API Benefits

Very easy to use

Minimizes memory footprint

Fuses and optimizes away graph nodes

Automatically handles padding

Automatically manages state objects

Training a Neural Network with MPS

Create training graph

Prepare inputs

Specify weights

Execute graph

- Graph updates weights

Complete training process

Prepare Inputs

Inputs to the graph

- Batch of source images
- Batch of source states

```
// Encode a batch of images for training
trainingGraph.encodeBatch(to: commandBuffer!,
                          sourceImages: [imageBatch],
                          sourceStates: [statesBatch])
```

Prepare Inputs

Inputs to the graph

- Batch of source images
- Batch of source states

```
// Encode a batch of images for training
trainingGraph.encodeBatch(to: commandBuffer!,
                          sourceImages: [imageBatch],
                          sourceStates: [statesBatch])
```

Batches

NEW

Batches are arrays of images or states

`MPSImageBatch`, `MPSStateBatch`

```
// Create an input image
let inputImage0 = MPSImage(texture: texture0, featureChannels: 4)
. . .

// Create a batch of input images
var imageBatch : [MPSImage] = []
imageBatch.append(inputImage0)
. . .
```

Batches

NEW

Batches are arrays of images or states

`MPSImageBatch`, `MPSStateBatch`

```
// Create an input image
let inputImage0 = MPSImage(texture: texture0, featureChannels: 4)
. . .

// Create a batch of input images
var imageBatch : [MPSImage] = []
imageBatch.append(inputImage0)
. . .
```

Batches

NEW

Batches are arrays of images or states

`MPSImageBatch`, `MPSStateBatch`

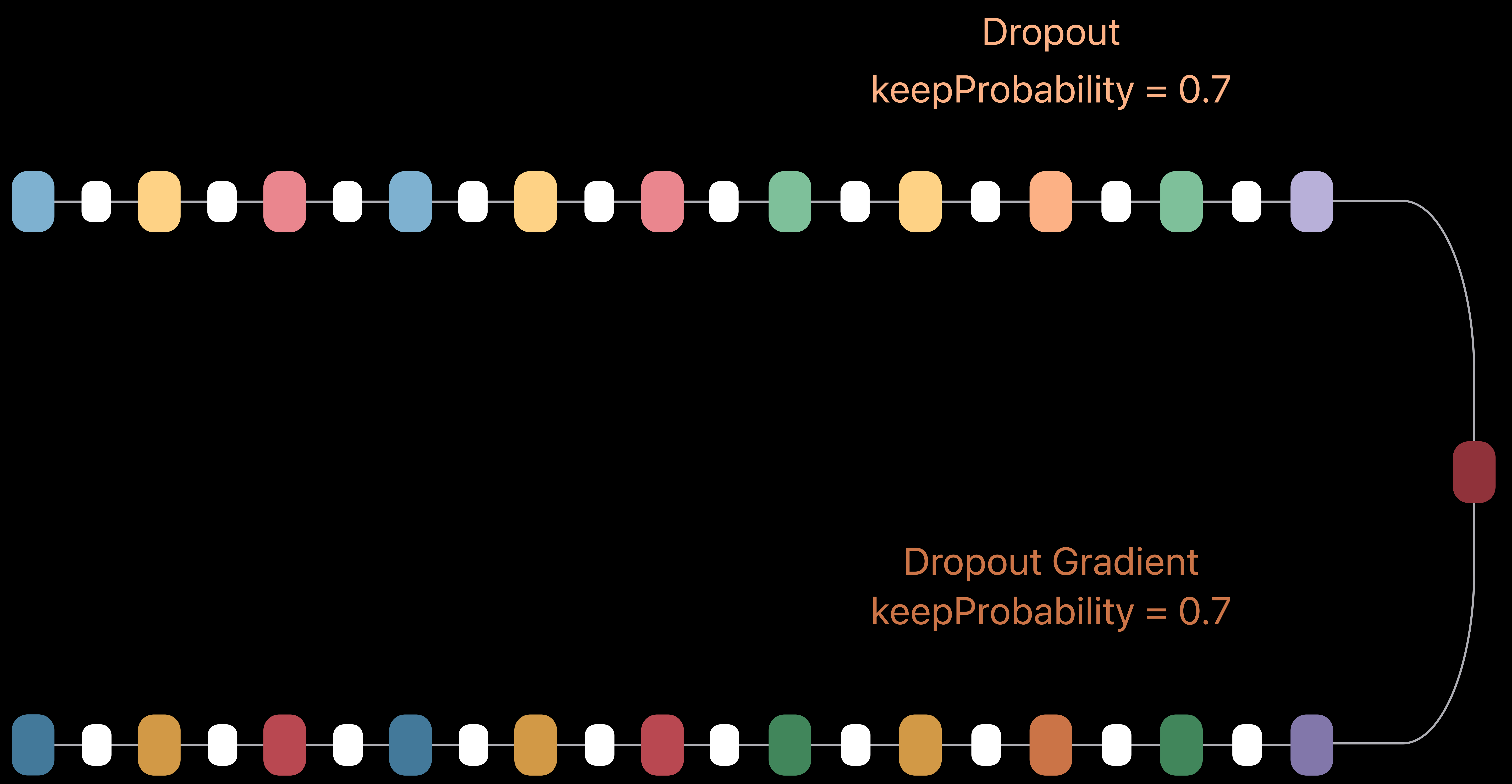
```
// Create an input image
let inputImage0 = MPSImage(texture: texture0, featureChannels: 4)
. . .

// Create a batch of input images
var imageBatch : [MPSImage] = []
imageBatch.append(inputImage0)
. . .
```

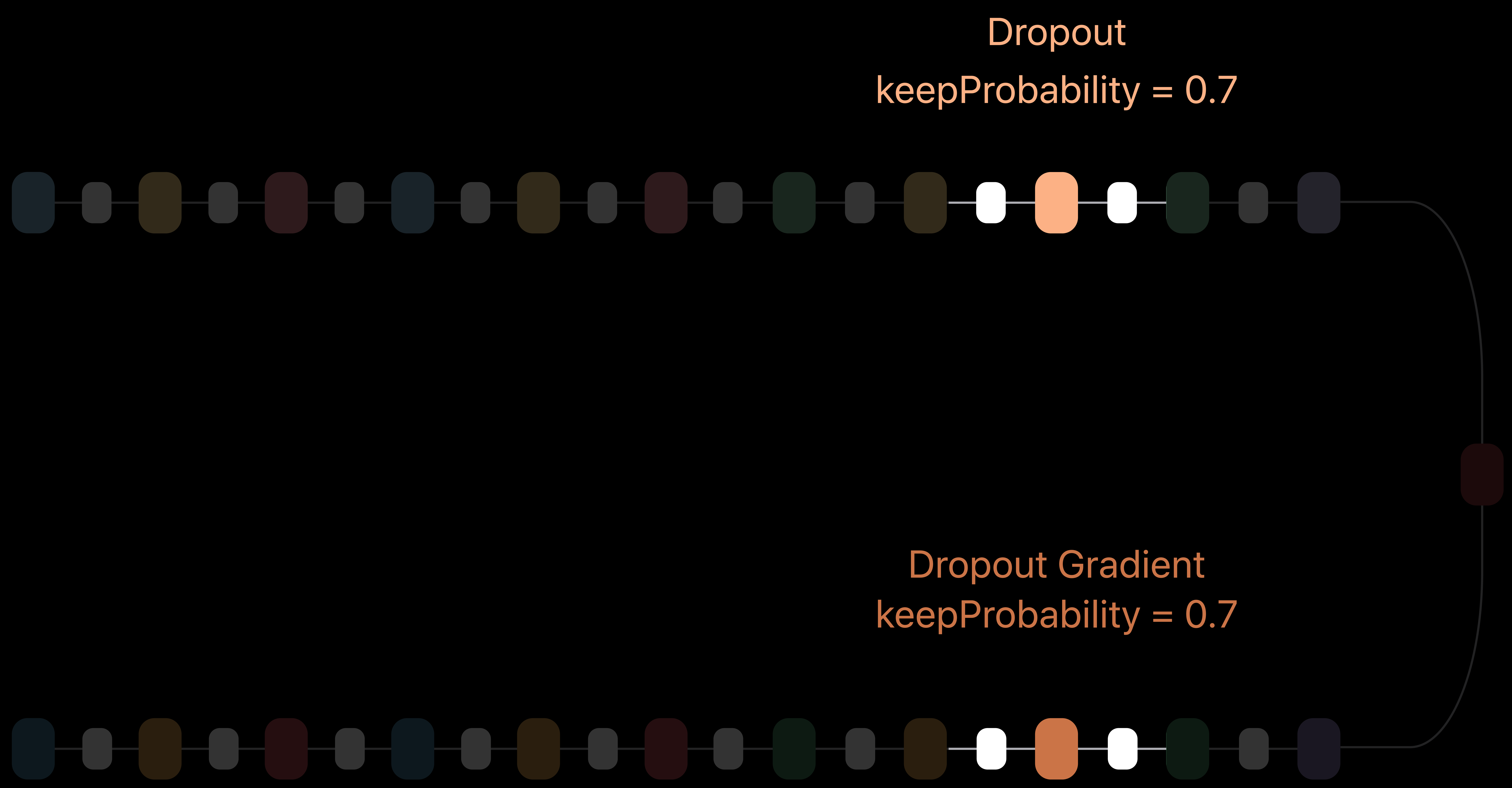
States

NEW

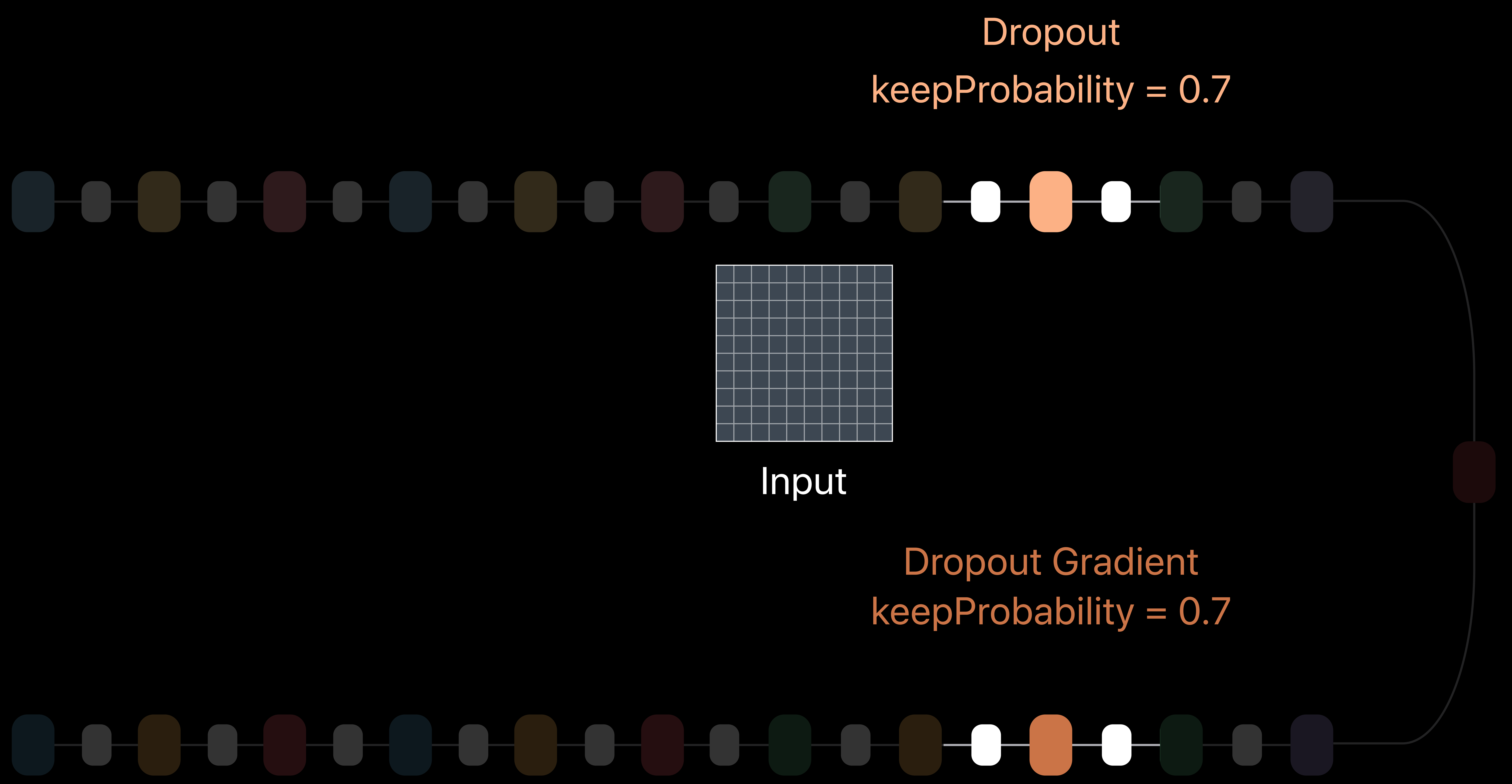
`MPSState` passes state of forward node to gradient node

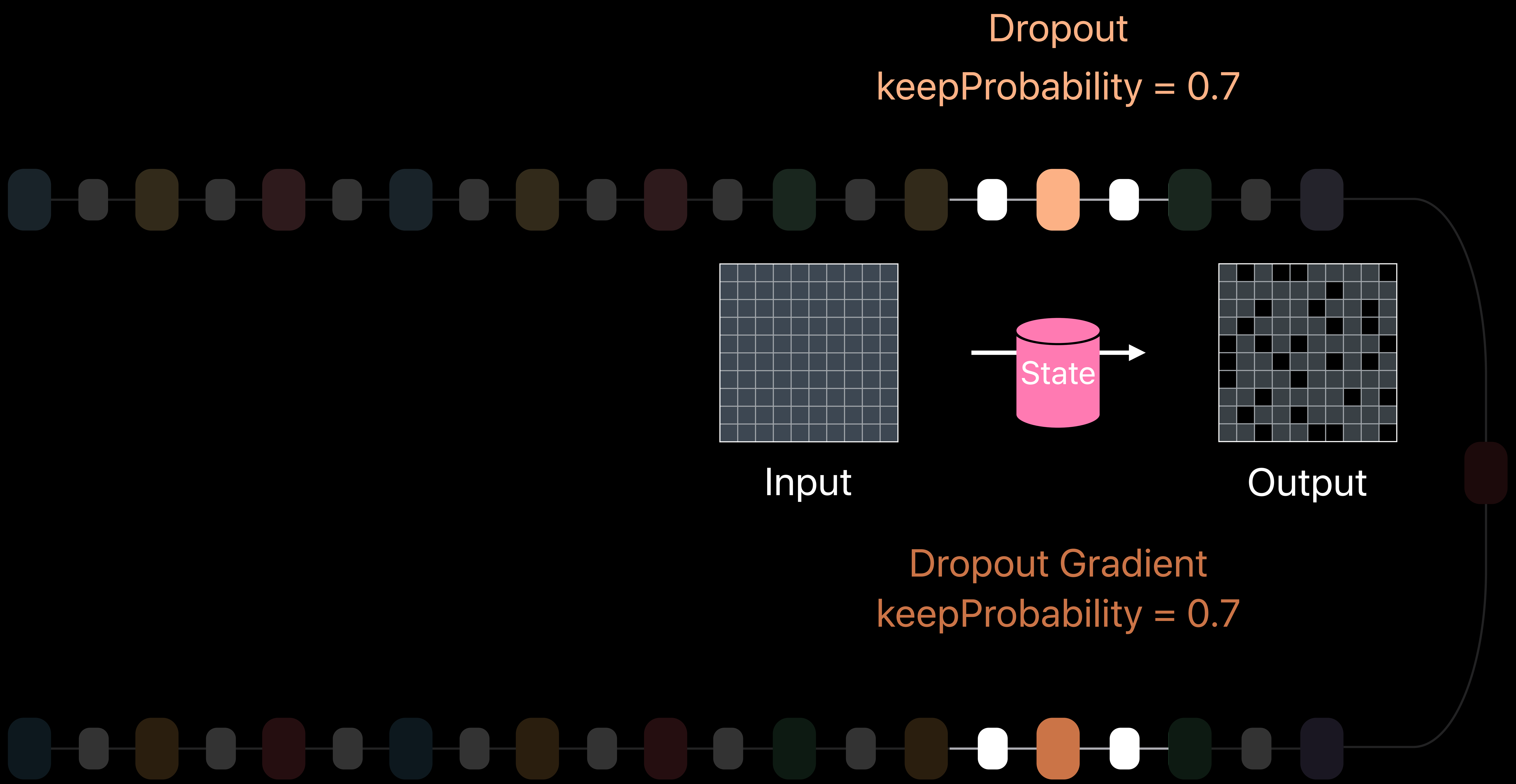
Graph manages all states

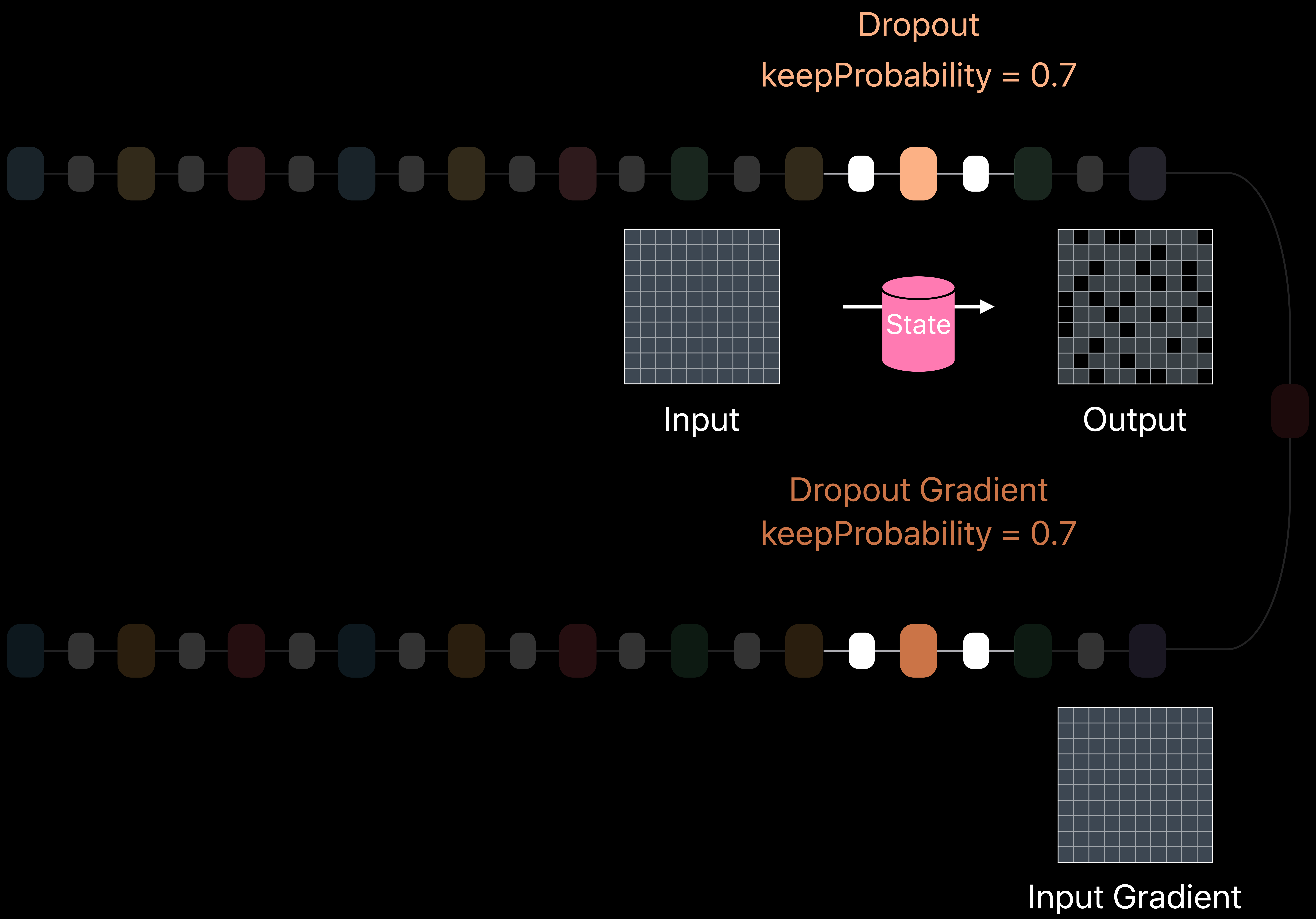

Dropout
keepProbability = 0.7
Dropout Gradient
keepProbability = 0.7



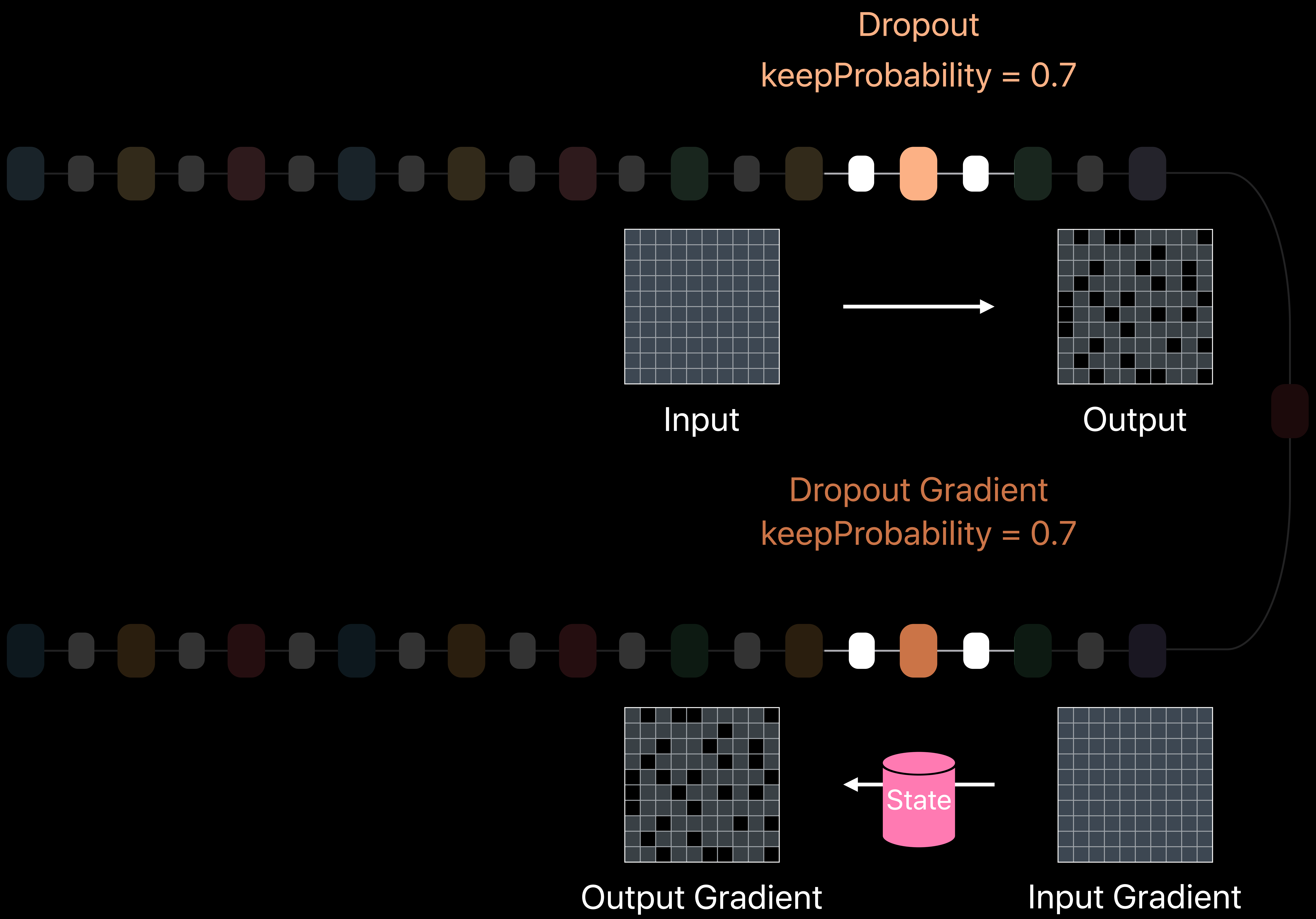
Dropout
keepProbability = 0.7
Dropout Gradient
keepProbability = 0.7


Dropout
keepProbability = 0.7
Input
Dropout Gradient
keepProbability = 0.7


Dropout
keepProbability = 0.7
State
Input
Output
Dropout Gradient
keepProbability = 0.7



Dropout
keepProbability = 0.7
State
Input
Output
Dropout Gradient
keepProbability = 0.7
Input Gradient


Dropout
keepProbability = 0.7
Input
Output
Dropout Gradient
keepProbability = 0.7
State
Output Gradient
Input Gradient

Loss Labels

```
// Create one label for Loss computation
let labelData = . . . // Load label data
let labelDesc = MPSCNNLossDataDescriptor(data: Data(bytes: labelData),
                                         layout: .featureChannelsxHeightxWidth,
                                         size: MTLSize(width: 1, height: 1, depth: 10))!

let cnnLabel = MPSCNNLossLabels(device: device,
                                labelsDescriptor: labelDesc)
```

Loss Labels

```swift
// Create one label for Loss computation
let labelData = . . . // Load label data
let labelDesc = MPSCNNLossDataDescriptor(data: Data(bytes: labelData),
                                         layout: .featureChannelsxHeightxWidth,
                                         size: MTLSize(width: 1, height: 1, depth: 10))!

let cnnLabel = MPSCNNLossLabels(device: device,
                                labelsDescriptor: labelDesc)
```

Loss Labels

```
// Create one label for Loss computation
let labelData = . . . // Load label data
let labelDesc = MPSCNNLossDataDescriptor(data: Data(bytes: labelData),
                                         layout: .featureChannelsxHeightxWidth,
                                         size: MTLSize(width: 1, height: 1, depth: 10))!

let cnnLabel = MPSCNNLossLabels(device: device,
                                labelsDescriptor: labelDesc)
```

Training a Neural Network with MPS

Create training graph

Prepare inputs

Specify weights

Execute graph

- Graph updates weights

Complete training process

Data Source Providers

Convolution

Fully Connected

Batch normalization

Instance normalization

```
// User implements MPSCNNConvolutionDataSource protocol
let conv = MPSCNNConvolutionNode(source: MPSNNImageNode(handle: nil),
                                 weights: MyWeights(withFile:"conv.dat"))
```

Data Source Providers

Convolution

Fully Connected

Batch normalization

Instance normalization

```
// User implements MPSCNNConvolutionDataSource protocol
let conv = MPSCNNConvolutionNode(source: MPSNNImageNode(handle: nil),
                                 weights: MyWeights(withFile:"conv.dat"))
```

Data Source Providers

Convolution

Fully Connected

Batch normalization

Instance normalization

```
// User implements MPSCNNConvolutionDataSource protocol
let conv = MPSCNNConvolutionNode(source: MPSNNImageNode(handle: nil),
                                 weights: MyWeights(withFile:"conv.dat"))
```

Data Source Providers

Just-in-time loading and purging of weights data

Minimize memory footprint

Data Source Providers

Just-in-time loading and purging of weights data

Minimize memory footprint

```swift
class MyWeights: NSObject, MPSCNNConvolutionDataSource {
   init(file: String) {…} // Initialize the data source object
   func load() -> Bool {…} // Graph calls load when weights for this layer are needed
   func descriptor() -> MPSCNNConvolutionDescriptor {…}
   func weights() -> UnsafeMutableRawPointer {…}
   func purge() {…} // When purge is called, you can release weights
}
```

Data Source Providers

Just-in-time loading and purging of weights data

Minimize memory footprint

```
class MyWeights: NSObject, MPSCNNConvolutionDataSource {
    init(file: String) {…} // Initialize the data source object
    func load() -> Bool {…} // Graph calls load when weights for this layer are needed
    func descriptor() -> MPSCNNConvolutionDescriptor {…}
    func weights() -> UnsafeMutableRawPointer {…}
    func purge() {…} // When purge is called, you can release weights
}
```

Data Source Providers

Just-in-time loading and purging of weights data

Minimize memory footprint

```
class MyWeights: NSObject, MPSCNNConvolutionDataSource {
    init(file: String) {…} // Initialize the data source object
    func load() -> Bool {…} // Graph calls load when weights for this layer are needed
    func descriptor() -> MPSCNNConvolutionDescriptor {…}
    func weights() -> UnsafeMutableRawPointer {…}
    func purge() {…} // When purge is called, you can release weights
}
```

Training a Neural Network with MPS

Create training graph

Prepare inputs

Specify weights

Execute graph

- Graph updates weights

Complete training process

```swift
// Example: Prepare to Execute Graph on the GPU
// Metal setup
let device = MTLCreateSystemDefaultDevice()

// Initialize graph
let trainingGraph = MPSNNGraph(device: device!, resultImage: makeTrainingGraph())

// Prepare inputs
. . .

// Train network on the GPU
. . .
```

```swift
// Example: Prepare to Execute Graph on the GPU
// Metal setup
let device = MTLCreateSystemDefaultDevice()

// Initialize graph
let trainingGraph = MPSNNGraph(device: device!, resultImage: makeTrainingGraph())

// Prepare inputs
. . .

// Train network on the GPU
. . .
```

```swift
// Example: Prepare to Execute Graph on the GPU
// Metal setup
let device = MTLCreateSystemDefaultDevice()

// Initialize graph
let trainingGraph = MPSNNGraph(device: device!, resultImage: makeTrainingGraph())

// Prepare inputs
. . .

// Train network on the GPU
. . .
```

```swift
// Example: Prepare to Execute Graph on the GPU
// Metal setup
let device = MTLCreateSystemDefaultDevice()

// Initialize graph
let trainingGraph = MPSNNGraph(device: device!, resultImage: makeTrainingGraph())

// Prepare inputs
. . .

// Train network on the GPU
. . .
```

```swift
// Example: Prepare to Execute Graph on the GPU
// Metal setup
let device = MTLCreateSystemDefaultDevice()

// Initialize graph
let trainingGraph = MPSNNGraph(device: device!, resultImage: makeTrainingGraph())

// Prepare inputs
. . .

// Train network on the GPU
. . .
```

```swift
// Example: Execute Graph in a Training Loop with Double Buffering

let latestCommandBuffer = nil

// NUM_EPOCHS is the number of times we iterate over an entire dataset
// NUM_ITERATIONS_PER_EPOCH is the number of images in a dataset, divided by batch size
for i in 0..<NUM_EPOCHS {
    for j in 0..<NUM_ITERATIONS_PER_EPOCH {
        latestCommandBuffer = trainingIteration(i * NUM_ITERATIONS_PER_EPOCH + j);
    }
    . . .
}
```

```swift
// Example: Execute Graph in a Training Loop with Double Buffering

let latestCommandBuffer = nil

// NUM_EPOCHS is the number of times we iterate over an entire dataset
// NUM_ITERATIONS_PER_EPOCH is the number of images in a dataset, divided by batch size
for i in 0..<NUM_EPOCHS {
    for j in 0..<NUM_ITERATIONS_PER_EPOCH {
        latestCommandBuffer = trainingIteration(i * NUM_ITERATIONS_PER_EPOCH + j);
    }
    . . .
}
```

```
// Example: Execute Graph in a Training Loop with Double Buffering

let latestCommandBuffer = nil

// NUM_EPOCHS is the number of times we iterate over an entire dataset
// NUM_ITERATIONS_PER_EPOCH is the number of images in a dataset, divided by batch size
for i in 0..<NUM_EPOCHS {
    for j in 0..<NUM_ITERATIONS_PER_EPOCH {
        latestCommandBuffer = trainingIteration(i * NUM_ITERATIONS_PER_EPOCH + j);
    }
    . . .
}
```

```swift
// Example: Execute Graph in a Training Loop with Double Buffering
let doubleBufferSemaphore = DispatchSemaphore(value: 2)

func trainingIteration(_ iter: Int) -> MTLCommandBuffer {
   doubleBufferSemaphore.wait(timeout: .distantFuture)
   let commandBuffer = commandQueue.makeCommandBuffer()!
   // Encode a batch of images for training
   var outputBatch = trainingGraph.encodeBatch(to: commandBuffer,
                                               sourceImages: [dataset.nextImageBatch(iter)],
                                               sourceStates: [dataset.nextLabelsBatch(iter)])
   commandBuffer.addCompletedHandler { commandBuffer in
      // Callback is called when GPU is done executing the graph (outputBatch is ready)
      doubleBufferSemaphore.signal()
   }

   commandBuffer.commit()
   return commandBuffer
}
```

```swift
// Example: Execute Graph in a Training Loop with Double Buffering
let doubleBufferSemaphore = DispatchSemaphore(value: 2)

func trainingIteration(_ iter: Int) -> MTLCommandBuffer {
    doubleBufferSemaphore.wait(timeout: .distantFuture)
    let commandBuffer = commandQueue.makeCommandBuffer()!
    // Encode a batch of images for training
    var outputBatch = trainingGraph.encodeBatch(to: commandBuffer,
                                                sourceImages: [dataset.nextImageBatch(iter)],
                                                sourceStates: [dataset.nextLabelsBatch(iter)])
    commandBuffer.addCompletedHandler { commandBuffer in
        // Callback is called when GPU is done executing the graph (outputBatch is ready)
        doubleBufferSemaphore.signal()
    }

    commandBuffer.commit()
    return commandBuffer
}
```

```swift
// Example: Execute Graph in a Training Loop with Double Buffering
let doubleBufferSemaphore = DispatchSemaphore(value: 2)

func trainingIteration(_ iter: Int) -> MTLCommandBuffer {
    doubleBufferSemaphore.wait(timeout: .distantFuture)
    let commandBuffer = commandQueue.makeCommandBuffer()!
    // Encode a batch of images for training
    var outputBatch = trainingGraph.encodeBatch(to: commandBuffer,
                                                 sourceImages: [dataset.nextImageBatch(iter)],
                                                 sourceStates: [dataset.nextLabelsBatch(iter)])
    commandBuffer.addCompletedHandler { commandBuffer in
        // Callback is called when GPU is done executing the graph (outputBatch is ready)
        doubleBufferSemaphore.signal()
    }

    commandBuffer.commit()
    return commandBuffer
}
```

```swift
// Example: Execute Graph in a Training Loop with Double Buffering
let doubleBufferSemaphore = DispatchSemaphore(value: 2)

func trainingIteration(_ iter: Int) -> MTLCommandBuffer {
    doubleBufferSemaphore.wait(timeout: .distantFuture)
    let commandBuffer = commandQueue.makeCommandBuffer()!
    // Encode a batch of images for training
    var outputBatch = trainingGraph.encodeBatch(to: commandBuffer,
                                                sourceImages: [dataset.nextImageBatch(iter)],
                                                sourceStates: [dataset.nextLabelsBatch(iter)])
    commandBuffer.addCompletedHandler { commandBuffer in
        // Callback is called when GPU is done executing the graph (outputBatch is ready)
        doubleBufferSemaphore.signal()
    }

    commandBuffer.commit()
    return commandBuffer
}
```

```swift
// Example: Execute Graph in a Training Loop with Double Buffering
let doubleBufferSemaphore = DispatchSemaphore(value: 2)

func trainingIteration(_ iter: Int) -> MTLCommandBuffer {
    doubleBufferSemaphore.wait(timeout: .distantFuture)
    let commandBuffer = commandQueue.makeCommandBuffer()!
    // Encode a batch of images for training
    var outputBatch = trainingGraph.encodeBatch(to: commandBuffer,
                                                sourceImages: [dataset.nextImageBatch(iter)],
                                                sourceStates: [dataset.nextLabelsBatch(iter)])
    commandBuffer.addCompletedHandler { commandBuffer in
        // Callback is called when GPU is done executing the graph (outputBatch is ready)
        doubleBufferSemaphore.signal()
    }

    commandBuffer.commit()
    return commandBuffer
}
```

Training a Neural Network with MPS

Create training graph

Prepare inputs

Specify weights

Execute graph

- Graph updates weights

Complete training process

Updating Weights

NEW

Implement optional update method on Data Source Provider

Graph calls update method automatically

```
class MyWeights: NSObject, MPSCNNConvolutionDataSource {
   . . .
   func update(with commandBuffer: MTLCommandBuffer,
                  gradientState: MPSCNNConvolutionGradientState,
                  sourceState: MPSCNNConvolutionWeightsAndBiasesState) ->
      MPSCNNConvolutionWeightsAndBiasesState? {…} // GPU
}
```

Updating Weights

NEW

Implement optional update method on Data Source Provider

Graph calls update method automatically

```
class MyWeights: NSObject, MPSCNNConvolutionDataSource {
    . . .
    func update(with commandBuffer: MTLCommandBuffer,
                gradientState: MPSCNNConvolutionGradientState,
                sourceState: MPSCNNConvolutionWeightsAndBiasesState) ->
       MPSCNNConvolutionWeightsAndBiasesState? {…} // GPU
}
```

Optimizers

For updating weights

NEW

Describe how to take update step on training parameters

Used in update method of Data Source Provider

Variants

- `MPSNNOptimizerAdam`
- `MPSNNOptimizerStochasticGradientDescent`
- `MPSNNOptimizerRMSProp`
- Custom

```swift
// Example: Use an Optimizer to Update Weights

// Add an optimizer to the init method of a Data Source Provider
init(kernelWidth: Int,
     kernelHeight: Int,
     inputFeatureChannels: Int,
     outputFeatureChannels: Int) {
   let length = kernelWidth * kernelHeight * inputFeatureChannels * outputFeatureChannels

   self.optimizer = MPSNNOptimizerStochasticGradientDescent(device: device,
                                                            learningRate: 0.01)
}
```

```swift
// Example: Use an Optimizer to Update Weights

// Add an optimizer to the init method of a Data Source Provider
init(kernelWidth: Int,
     kernelHeight: Int,
     inputFeatureChannels: Int,
     outputFeatureChannels: Int) {
   let length = kernelWidth * kernelHeight * inputFeatureChannels * outputFeatureChannels

   self.optimizer = MPSNNOptimizerStochasticGradientDescent(device: device,
                                                             learningRate: 0.01)
}
```

```swift
// Example: Use an Optimizer to Update Weights

func update(with commandBuffer: MTLCommandBuffer,
            gradientState: MPSCNNConvolutionGradientState,
            sourceState: MPSCNNConvolutionWeightsAndBiasesState)
   -> MPSCNNConvolutionWeightsAndBiasesState? {

   optimizer.encode(commandBuffer: commandBuffer,
                    convolutionGradientState: gradientState,
                    convolutionSourceState: sourceState,
                    inputMomentumVectors: nil,
                    resultState: sourceState)

   return sourceState
}
```

```swift
// Example: Use an Optimizer to Update Weights

func update(with commandBuffer: MTLCommandBuffer,
            gradientState: MPSCNNConvolutionGradientState,
            sourceState: MPSCNNConvolutionWeightsAndBiasesState)
    -> MPSCNNConvolutionWeightsAndBiasesState? {

    optimizer.encode(commandBuffer: commandBuffer,
                     convolutionGradientState: gradientState,
                     convolutionSourceState: sourceState,
                     inputMomentumVectors: nil,
                     resultState: sourceState)

    return sourceState
}
```

```swift
// Example: Use an Optimizer to Update Weights

func update(with commandBuffer: MTLCommandBuffer,
            gradientState: MPSCNNConvolutionGradientState,
            sourceState: MPSCNNConvolutionWeightsAndBiasesState)
   -> MPSCNNConvolutionWeightsAndBiasesState? {

    optimizer.encode(commandBuffer: commandBuffer,
                     convolutionGradientState: gradientState,
                     convolutionSourceState: sourceState,
                     inputMomentumVectors: nil,
                     resultState: sourceState)

    return sourceState
}
```

```swift
// Example: Use an Optimizer to Update Weights

func update(with commandBuffer: MTLCommandBuffer,
            gradientState: MPSCNNConvolutionGradientState,
            sourceState: MPSCNNConvolutionWeightsAndBiasesState)
   -> MPSCNNConvolutionWeightsAndBiasesState? {

   optimizer.encode(commandBuffer: commandBuffer,
                    convolutionGradientState: gradientState,
                    convolutionSourceState: sourceState,
                    inputMomentumVectors: nil,
                    resultState: sourceState)

   return sourceState
}
```

Training a Neural Network with MPS

Create training graph

Prepare inputs

Specify weights

Execute graph

- Graph updates weights

Complete training process

```swift
// Example: Training Loop with Double Buffering

let latestCommandBuffer = nil

// NUM_EPOCHS is the number of times we iterate over an entire dataset
// NUM_ITERATIONS_PER_EPOCH is the number of images in a dataset, divided by batch size
for i in 0..<NUM_EPOCHS {
    for j in 0..<NUM_ITERATIONS_PER_EPOCH {
        latestCommandBuffer = trainingIteration(i * NUM_ITERATIONS_PER_EPOCH + j);
    }

    // Optional steps after each epoch:
    latestCommandBuffer.waitUntilCompleted()
    // Run inference with current trained parameters on a test set
    // Optionally, stop training when the network reaches an acceptable level of accuracy
}
```

```
// Example: Training Loop with Double Buffering

let latestCommandBuffer = nil

// NUM_EPOCHS is the number of times we iterate over an entire dataset
// NUM_ITERATIONS_PER_EPOCH is the number of images in a dataset, divided by batch size
for i in 0..<NUM_EPOCHS {
    for j in 0..<NUM_ITERATIONS_PER_EPOCH {
        latestCommandBuffer = trainingIteration(i * NUM_ITERATIONS_PER_EPOCH + j);
    }

    // Optional steps after each epoch:
    latestCommandBuffer.waitUntilCompleted()
    // Run inference with current trained parameters on a test set
    // Optionally, stop training when the network reaches an acceptable level of accuracy
}
```

```
// Example: Training Loop with Double Buffering

let latestCommandBuffer = nil

// NUM_EPOCHS is the number of times we iterate over an entire dataset
// NUM_ITERATIONS_PER_EPOCH is the number of images in a dataset, divided by batch size
for i in 0..<NUM_EPOCHS {
    for j in 0..<NUM_ITERATIONS_PER_EPOCH {
        latestCommandBuffer = trainingIteration(i * NUM_ITERATIONS_PER_EPOCH + j);
    }

    // Optional steps after each epoch:
    latestCommandBuffer.waitUntilCompleted()
    // Run inference with current trained parameters on a test set
    // Optionally, stop training when the network reaches an acceptable level of accuracy
}
```

Demo

Object detection training using Turi Create with MPS

RNN Training

CNN

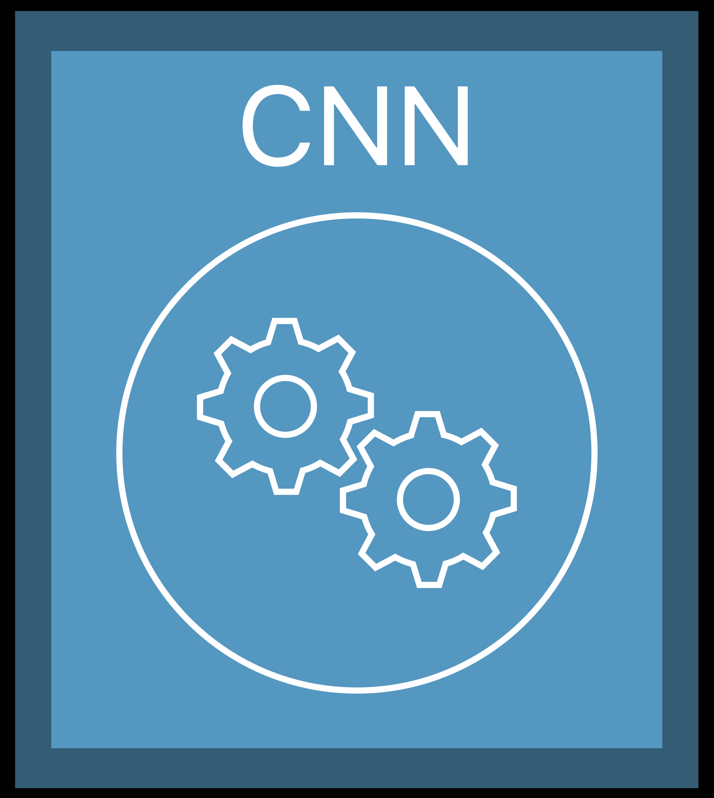
One-to-one

One image

CNN

One-to-one

One image

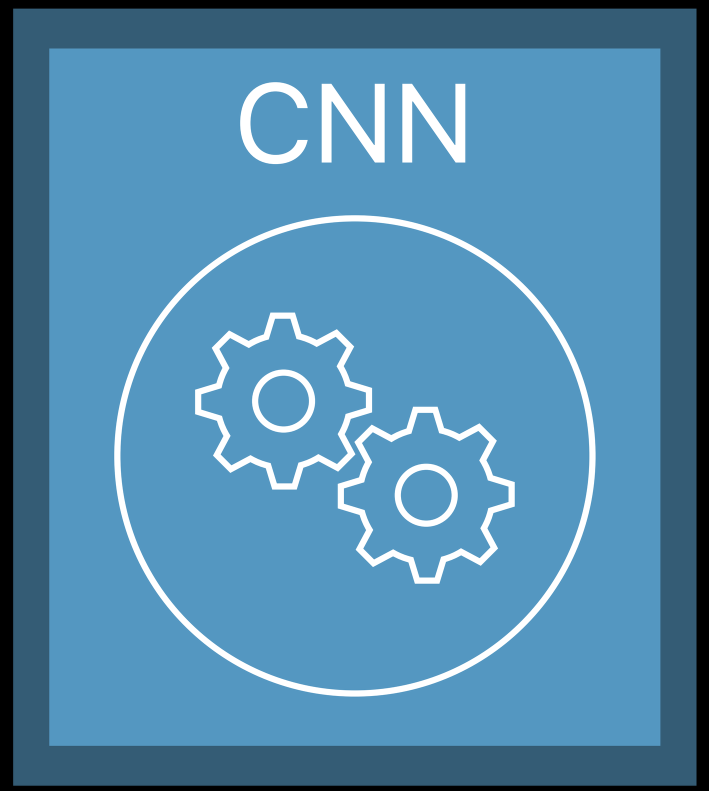


cat
dog
grass
…

One set of probabilities

RNN

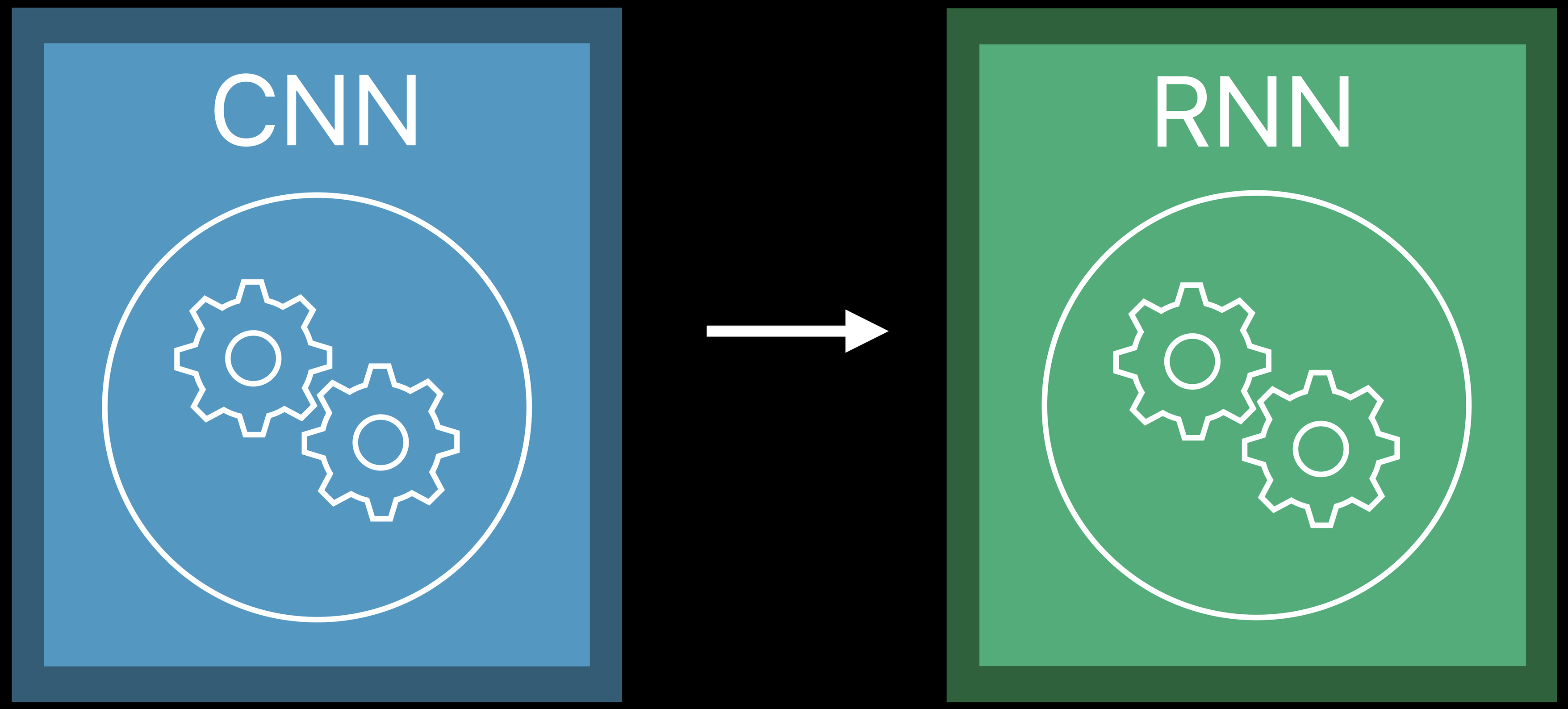
Sequences: one-to-many



One image

RNN

Sequences: one-to-many



A dog is laying on the grass next to a cat

One image

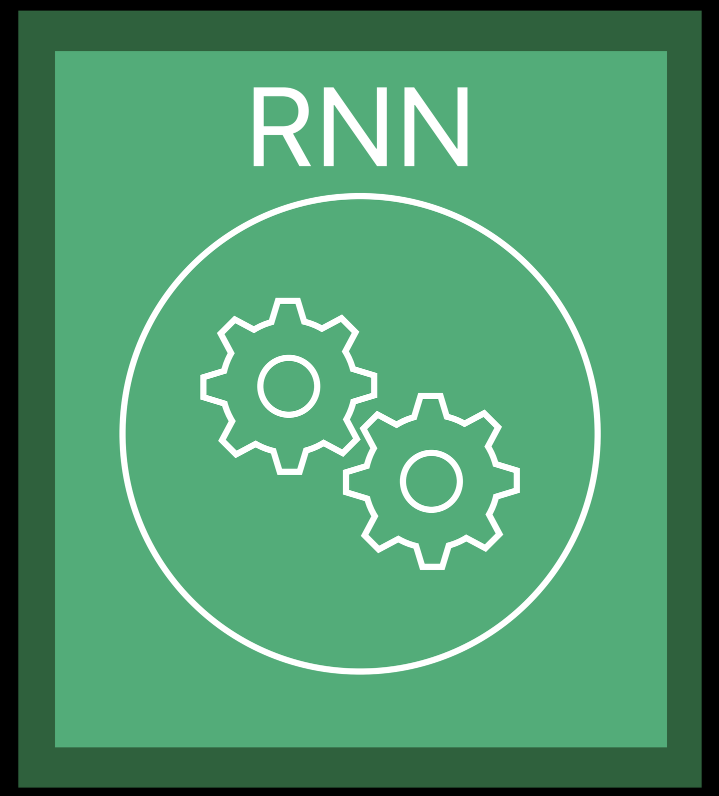
One set of probabilities

Sequence of words

RNN

Sequences: many-to-many

A dog is laying
on the grass
next to a cat

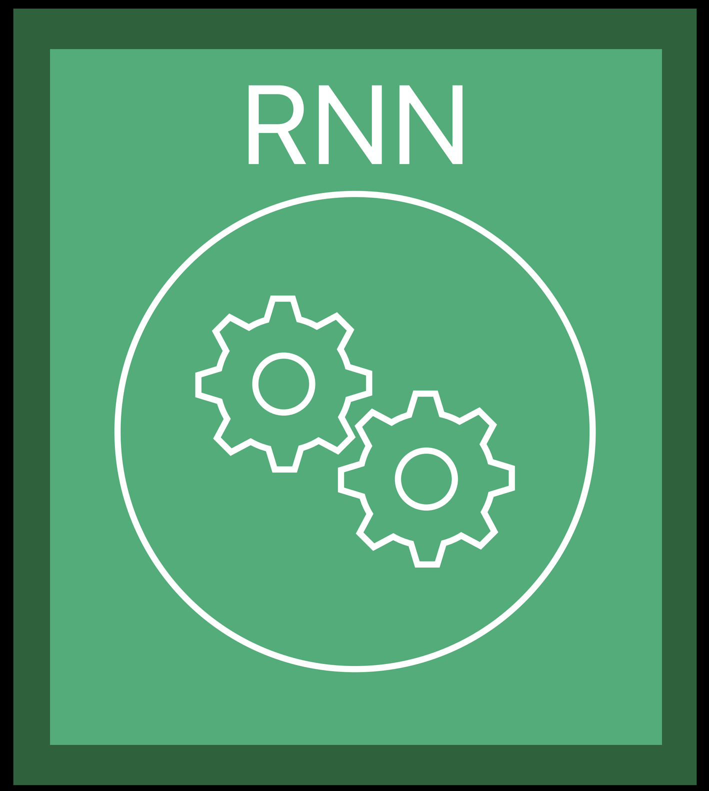


Sequence
of words

RNN

Sequences: many-to-many

A dog is laying
on the grass
next to a cat



Собака
лежит на
траве рядом
с котом

Koira makaa
ruohikolla
kissan
vieressä

Sequence
of words

Sequence
of words

Recurrent Neural Networks

Variants for inference

- Single Gate
- Long Short-Term Memory (LSTM)
- Gated Recurrent Unit (GRU)
- Minimally Gated Unit (MGU)

Recurrent Neural Networks

NEW

Variants for inference and training

- Single Gate
- Long Short-Term Memory (LSTM)
- Gated Recurrent Unit (GRU)
- Minimally Gated Unit (MGU)

Activity Classifier
Network overview

*Graphical representation of motion-sensory data for the purpose of demonstration, not actual data

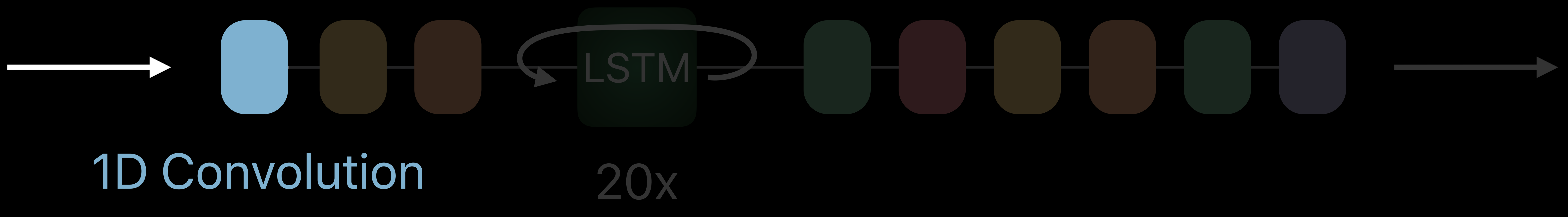
Inference


Convolution
ReLu
Fully Connected
Dropout
Batch Normalization
SoftMax
LSTM
1D images with
2000 pixels
(samples in time)
LSTM
20x
Activity
Predictions
Input
Output

Inference

Convolution
ReLu
Fully Connected
Dropout
Batch Normalization
SoftMax
LSTM

1D images with 2000 pixels (samples in time)



1D Convolution

"Compresses" input to 20 samples

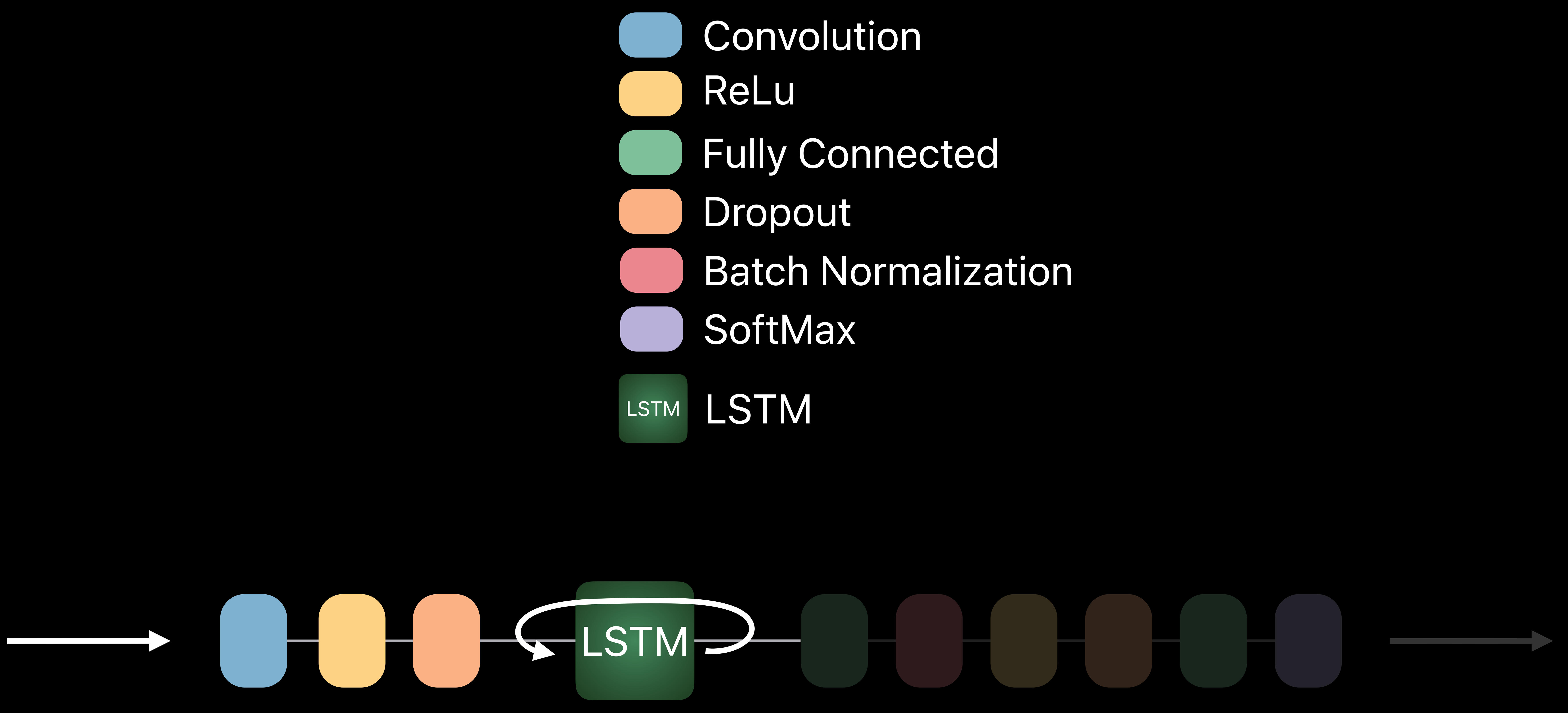
Activity Predictions

Input

Output

Inference

1D images with
2000 pixels
(samples in time)



20x

Activity
Predictions

“Compresses”
input to 20
samples

Sequence
of length 20

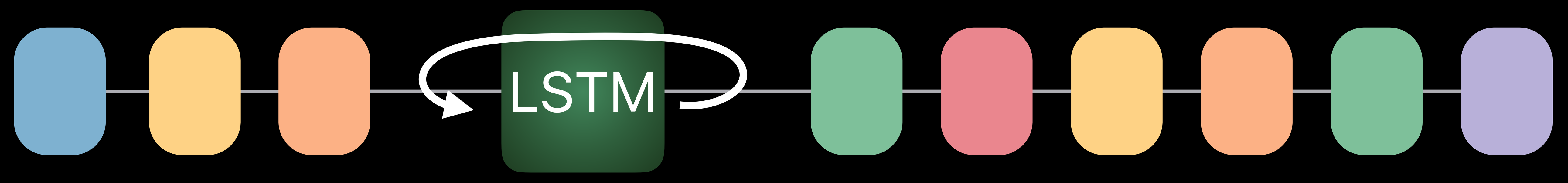
Input

Output

Inference

Convolution
ReLu
Fully Connected
Dropout
Batch Normalization
SoftMax
LSTM

1D images with 2000 pixels (samples in time)



20x

Activity Predictions

"Compresses" input to 20 samples

Sequence of length 20

Refine higher level features

Input

Output

Training


Convolution
ReLu
Fully Connected
Dropout
Batch Normalization
SoftMax
LSTM
LSTM
Loss
Convolution Gradient
ReLu Gradient
Fully Connected Gradient
Dropout Gradient
Batch Normalization Gradient
SoftMax Gradient
LSTM
LSTM Gradient
Images
LSTM
20x
Activity Predictions
Labels

Training


Convolution
ReLu
Fully Connected
Dropout
Batch Normalization
SoftMax
LSTM
LSTM
Loss
Convolution Gradient
ReLu Gradient
Fully Connected Gradient
Dropout Gradient
Batch Normalization Gradient
SoftMax Gradient
LSTM
LSTM Gradient
Images
LSTM
20x
Activity Predictions
Labels
LSTM
20x

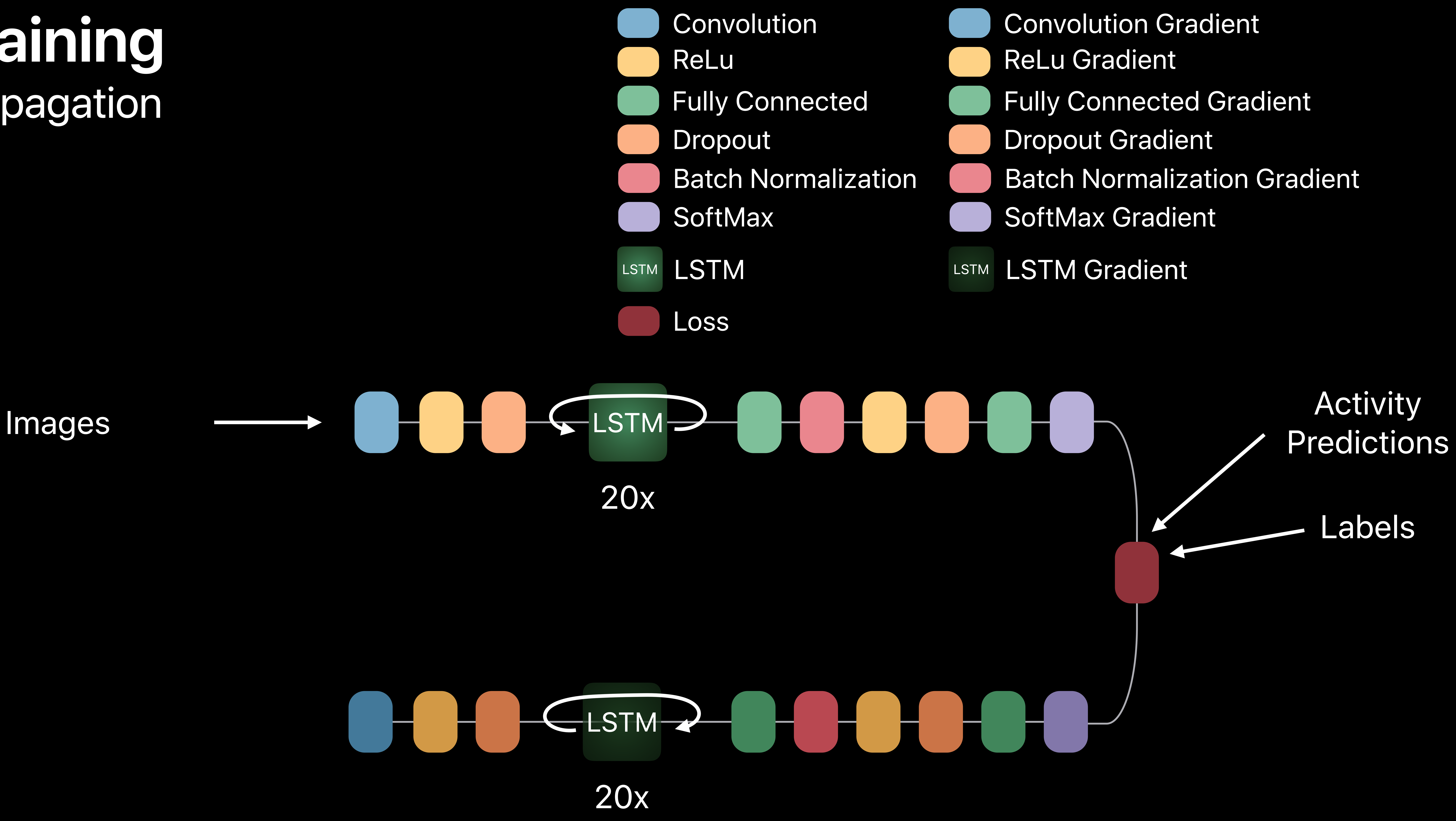
Training
Propagation

- Convolution
- ReLu
- Fully Connected
- Dropout
- Batch Normalization
- SoftMax
- LSTM
- Loss

- Convolution Gradient
- ReLu Gradient
- Fully Connected Gradient
- Dropout Gradient
- Batch Normalization Gradient
- SoftMax Gradient
- LSTM Gradient

Training

Propagation



Training
Weights update

Training

Weights update

```swift
// Example: Create a LSTM Layer for Training

// Create a LSTM layer descriptor
// Initialize descriptor with initial training parameters, using data source providers
// Descriptor setup is exactly the same as for inference
let descriptor = MPSLSTMDescriptor()
descriptor.useFloat32Weights = true

// Create a LSTM layer for training
let trainingWeights: NSMutableArray = [] // matrices to hold training parameters
let weightGradients: NSMutableArray = [] // matrices to hold gradients
let trainingLayer = MPSRNNMatrixTrainingLayer(device: device!, rnnDescriptor: descriptor,
                                              trainableWeights: trainingWeights)
trainingLayer.createWeightGradientMatrices(weightGradients, dataType: .float32)
```

```swift
// Example: Create a LSTM Layer for Training

// Create a LSTM layer descriptor
// Initialize descriptor with initial training parameters, using data source providers
// Descriptor setup is exactly the same as for inference
let descriptor = MPSLSTMDescriptor()
descriptor.useFloat32Weights = true

// Create a LSTM layer for training
let trainingWeights: NSMutableArray = [] // matrices to hold training parameters
let weightGradients: NSMutableArray = [] // matrices to hold gradients
let trainingLayer = MPSRNNMatrixTrainingLayer(device: device!, rnnDescriptor: descriptor,
                                              trainableWeights: trainingWeights)
trainingLayer.createWeightGradientMatrices(weightGradients, dataType: .float32)
```

```swift
// Example: Create a LSTM Layer for Training

// Create a LSTM layer descriptor
// Initialize descriptor with initial training parameters, using data source providers
// Descriptor setup is exactly the same as for inference
let descriptor = MPSLSTMDescriptor()
descriptor.useFloat32Weights = true

// Create a LSTM layer for training
let trainingWeights: NSMutableArray = [] // matrices to hold training parameters
let weightGradients: NSMutableArray = [] // matrices to hold gradients
let trainingLayer = MPSRNNMatrixTrainingLayer(device: device!, rnnDescriptor: descriptor,
                                              trainableWeights: trainingWeights)
trainingLayer.createWeightGradientMatrices(weightGradients, dataType: .float32)
```

```swift
// Example: Create a LSTM Layer for Training

// Create a LSTM layer descriptor
// Initialize descriptor with initial training parameters, using data source providers
// Descriptor setup is exactly the same as for inference
let descriptor = MPSLSTMDescriptor()
descriptor.useFloat32Weights = true

// Create a LSTM layer for training
let trainingWeights: NSMutableArray = [] // matrices to hold training parameters
let weightGradients: NSMutableArray = [] // matrices to hold gradients
let trainingLayer = MPSRNNMatrixTrainingLayer(device: device!, rnnDescriptor: descriptor,
                                              trainableWeights: trainingWeights)
trainingLayer.createWeightGradientMatrices(weightGradients, dataType: .float32)
```

```swift
// Example: Create a LSTM Layer for Training

// Create a LSTM layer descriptor
// Initialize descriptor with initial training parameters, using data source providers
// Descriptor setup is exactly the same as for inference
let descriptor = MPSLSTMDescriptor()
descriptor.useFloat32Weights = true

// Create a LSTM layer for training
let trainingWeights: NSMutableArray = [] // matrices to hold training parameters
let weightGradients: NSMutableArray = [] // matrices to hold gradients
let trainingLayer = MPSRNNMatrixTrainingLayer(device: device!, rnnDescriptor: descriptor,
                                              trainableWeights: trainingWeights)
trainingLayer.createWeightGradientMatrices(weightGradients, dataType: .float32)
```

```swift
// Example: Create a LSTM Layer for Training

// Create a LSTM layer descriptor
// Initialize descriptor with initial training parameters, using data source providers
// Descriptor setup is exactly the same as for inference
let descriptor = MPSLSTMDescriptor()
descriptor.useFloat32Weights = true

// Create a LSTM layer for training
let trainingWeights: NSMutableArray = [] // matrices to hold training parameters
let weightGradients: NSMutableArray = [] // matrices to hold gradients
let trainingLayer = MPSRNNMatrixTrainingLayer(device: device!, rnnDescriptor: descriptor,
                                              trainableWeights: trainingWeights)
trainingLayer.createWeightGradientMatrices(weightGradients, dataType: .float32)
```

```swift
// Example: Prepare Inputs and Outputs for Training a LSTM

// Create 20 matrices to hold input and output sequences for forward and gradient passes
var inputSequence: [MPSMatrix] = []
var outputSequence: [MPSMatrix] = []
var inputGradientSequence: [MPSMatrix] = []
var outputGradientSequence: [MPSMatrix] = []

for i in 0..< 20 {
   // Matrix size is (1, inputSize), inputSize is number of columns
   inputSequence.append(MPSMatrix(…))
   // Initialize matrices for the other sequences
   . . .
}
```

```swift
// Example: Prepare Inputs and Outputs for Training a LSTM

// Create 20 matrices to hold input and output sequences for forward and gradient passes
var inputSequence: [MPSMatrix] = []
var outputSequence: [MPSMatrix] = []
var inputGradientSequence: [MPSMatrix] = []
var outputGradientSequence: [MPSMatrix] = []

for i in 0..< 20 {
    // Matrix size is (1, inputSize), inputSize is number of columns
    inputSequence.append(MPSMatrix(…))
    // Initialize matrices for the other sequences
    . . .
}
```

```swift
// Example: Prepare Inputs and Outputs for Training a LSTM

// Create 20 matrices to hold input and output sequences for forward and gradient passes
var inputSequence: [MPSMatrix] = []
var outputSequence: [MPSMatrix] = []
var inputGradientSequence: [MPSMatrix] = []
var outputGradientSequence: [MPSMatrix] = []

for i in 0..< 20 {
    // Matrix size is (1, inputSize), inputSize is number of columns
    inputSequence.append(MPSMatrix(…))
    // Initialize matrices for the other sequences
    . . .
}
```

```swift
// Example: Train Activity Classifier Network with MPS

// Run CNN kernels in forward pass

let trainingStates: NSMutableArray = [] // States pass data from forward to gradient pass
// Run sequence of 20 matrices through LSTM training layer (forward pass)
trainingLayer.encodeForwardSequence(commandBuffer: commandBuffer,
                                    sourceMatrices: inputSequence,
                                    destinationMatrices: outputSequence,
                                    trainingStates: trainingStates,
                                    weights: trainingWeights)

// Run additional CNN kernels in forward pass
// Compute Loss
```

```swift
// Example: Train Activity Classifier Network with MPS

// Run CNN kernels in forward pass

let trainingStates: NSMutableArray = [] // States pass data from forward to gradient pass
// Run sequence of 20 matrices through LSTM training layer (forward pass)
trainingLayer.encodeForwardSequence(commandBuffer: commandBuffer,
                                    sourceMatrices: inputSequence,
                                    destinationMatrices: outputSequence,
                                    trainingStates: trainingStates,
                                    weights: trainingWeights)

// Run additional CNN kernels in forward pass
// Compute Loss
```

```swift
// Example: Train Activity Classifier Network with MPS, cont.

// Run CNN kernels in gradient pass

// Run sequence of 20 matrices through LSTM training layer (gradient pass)
trainingLayer.encodeGradientSequence(commandBuffer: commandBuffer,
                                     forwardSources: forwardSources,
                                     sourceGradients: inputGradientSequence,
                                     destinationGradients: outputGradientSequence,
                                     weightGradients: weightGradients,
                                     trainingStates: trainingStates,
                                     weights: trainingWeights)

// Run additional CNN kernels in gradient pass
// Use MPSNNOptimizer to update trainingWeights using weightGradients
// Submit work to the GPU
```

```swift
// Example: Train Activity Classifier Network with MPS, cont.

// Run CNN kernels in gradient pass

// Run sequence of 20 matrices through LSTM training layer (gradient pass)
trainingLayer.encodeGradientSequence(commandBuffer: commandBuffer,
                                     forwardSources: forwardSources,
                                     sourceGradients: inputGradientSequence,
                                     destinationGradients: outputGradientSequence,
                                     weightGradients: weightGradients,
                                     trainingStates: trainingStates,
                                     weights: trainingWeights)

// Run additional CNN kernels in gradient pass
// Use MPSNNOptimizer to update trainingWeights using weightGradients
// Submit work to the GPU
```

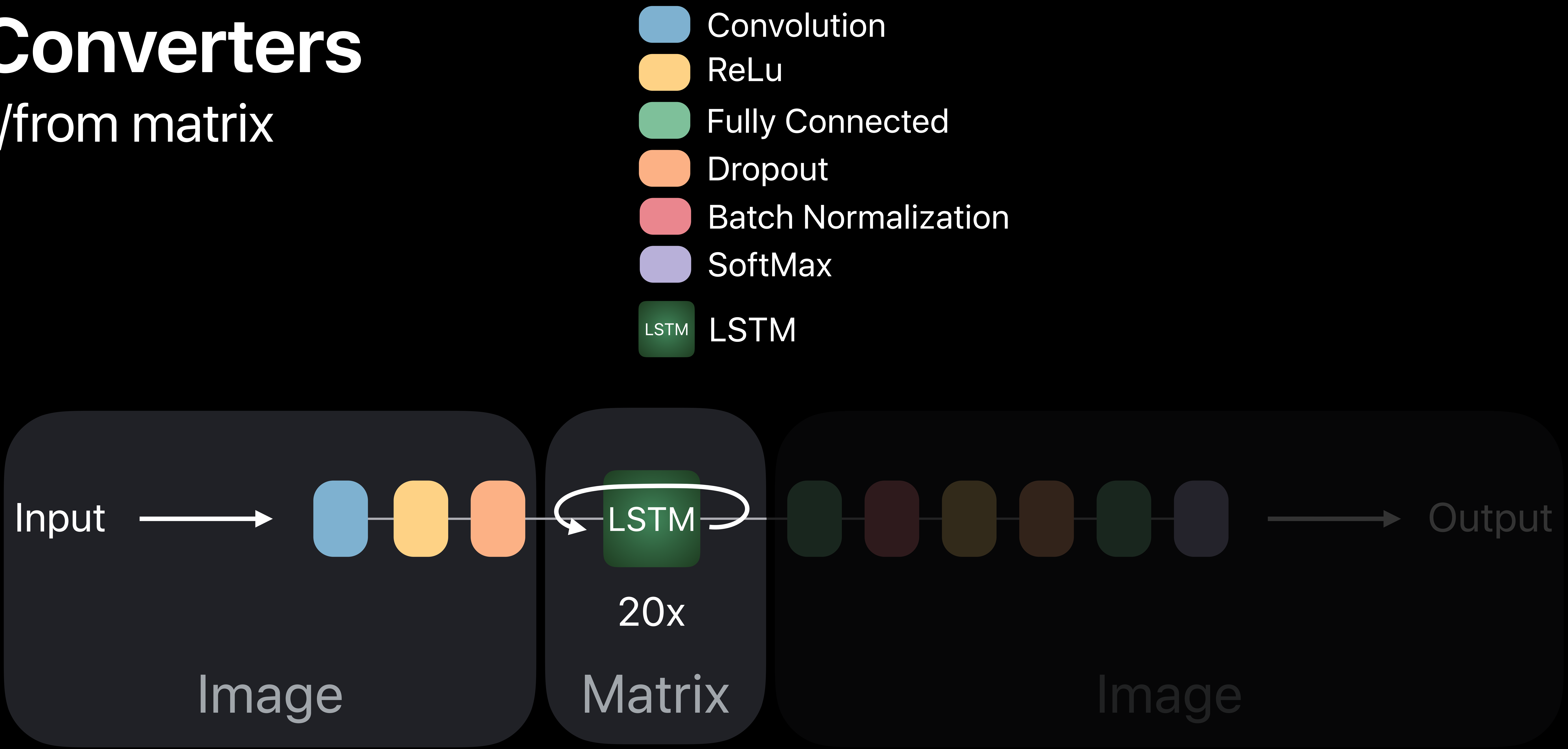
```swift
// Example: Train Activity Classifier Network with MPS, cont.

// Run CNN kernels in gradient pass

// Run sequence of 20 matrices through LSTM training layer (gradient pass)
trainingLayer.encodeGradientSequence(commandBuffer: commandBuffer,
                                     forwardSources: forwardSources,
                                     sourceGradients: inputGradientSequence,
                                     destinationGradients: outputGradientSequence,
                                     weightGradients: weightGradients,
                                     trainingStates: trainingStates,
                                     weights: trainingWeights)

// Run additional CNN kernels in gradient pass
// Use MPSNNOptimizer to update trainingWeights using weightGradients
// Submit work to the GPU
```

Data Converters

Image to/from matrix

Data Converters

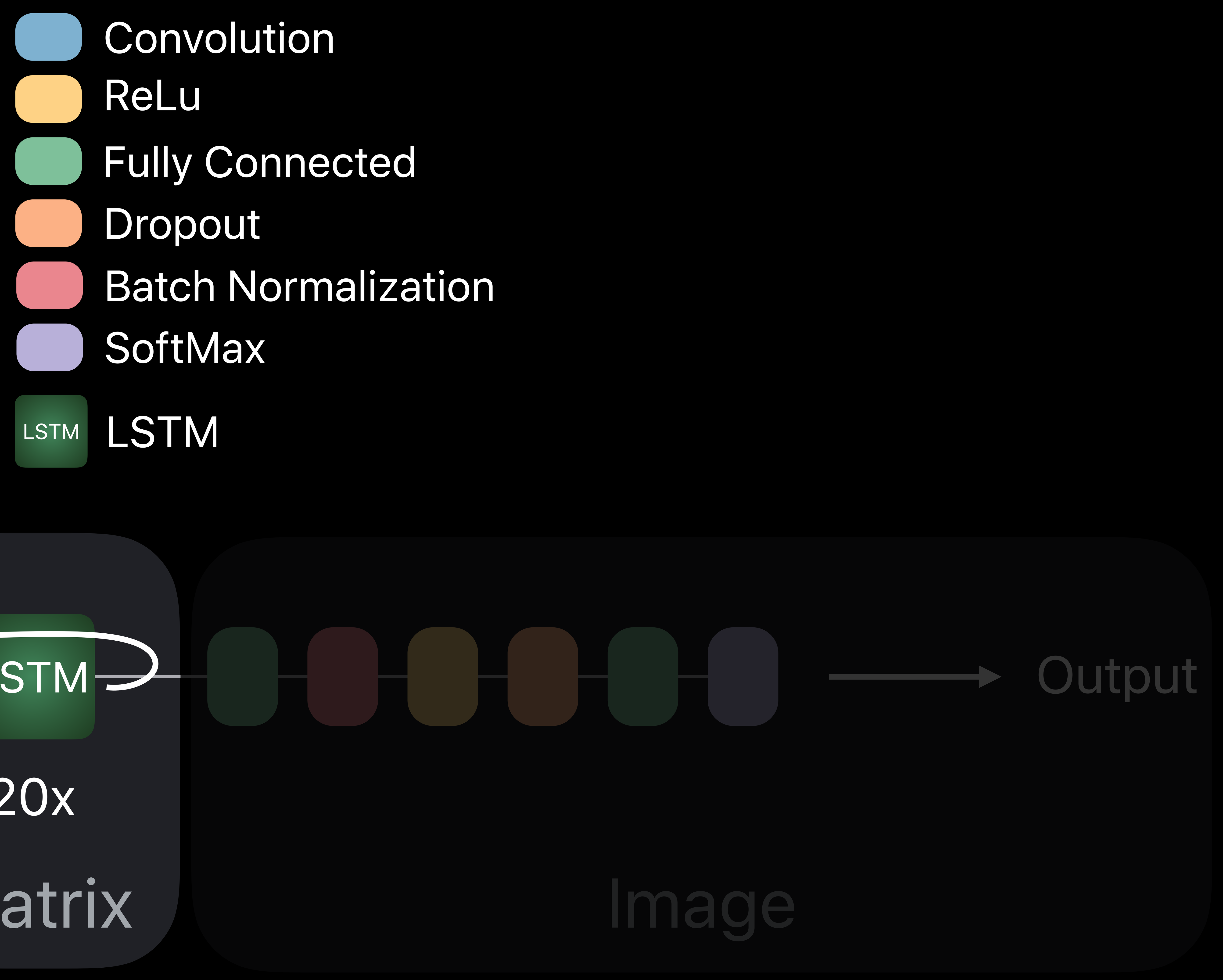
Image to/from matrix




```
let imageToMatrix = MPSImageCopyToMatrix(device: device, dataLayout: .featureChannelsxHeightxWidth)
imageToMatrix.encodeBatch(commandBuffer: commandBuffer, sourceImages: [image1, image2],
                          destinationMatrix: matrix)
```

Data Converters

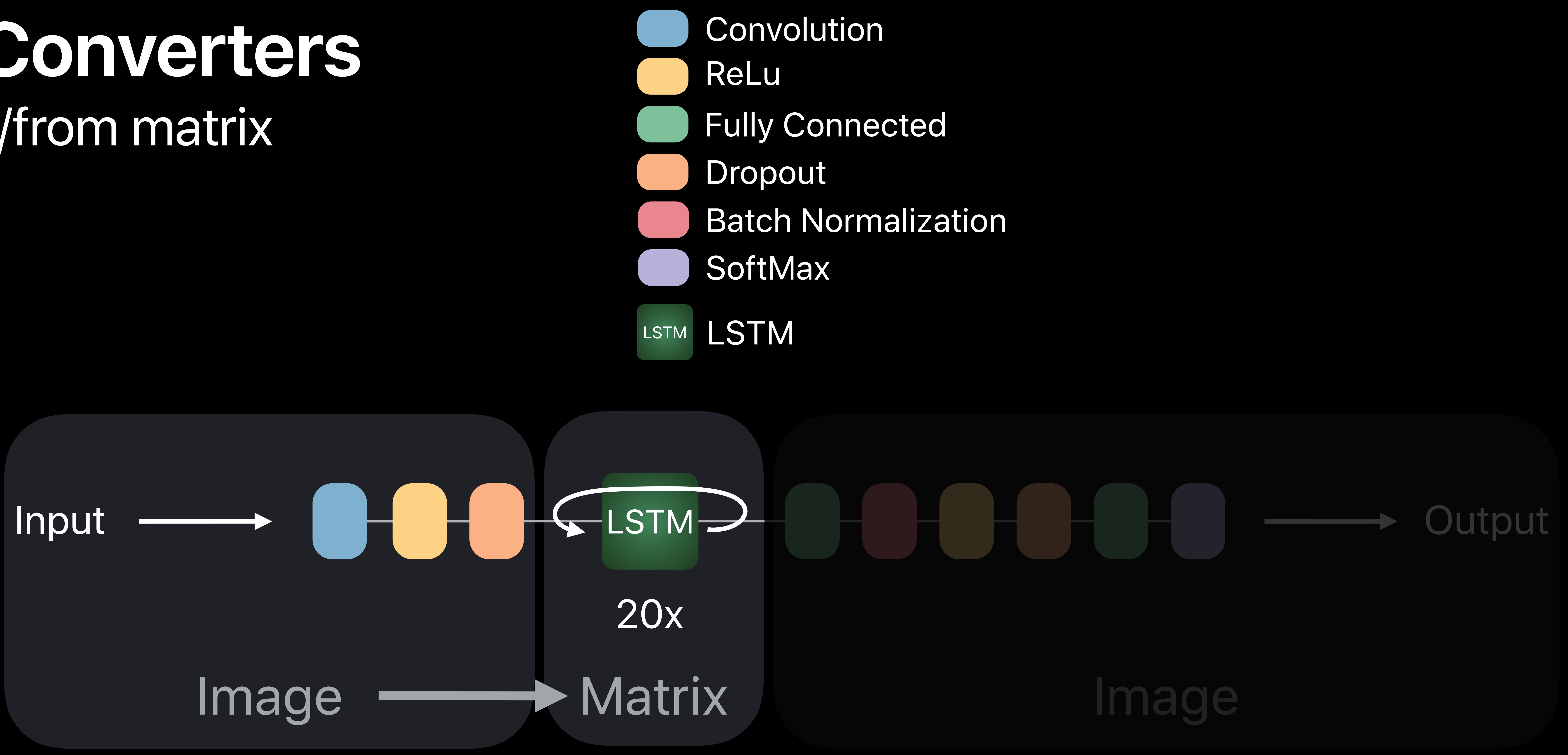
Image to/from matrix




```
let imageToMatrix = MPSImageCopyToMatrix(device: device, dataLayout: .featureChannelsxHeightxWidth)
imageToMatrix.encodeBatch(commandBuffer: commandBuffer, sourceImages: [image1, image2],
                          destinationMatrix: matrix)
```

Data Converters

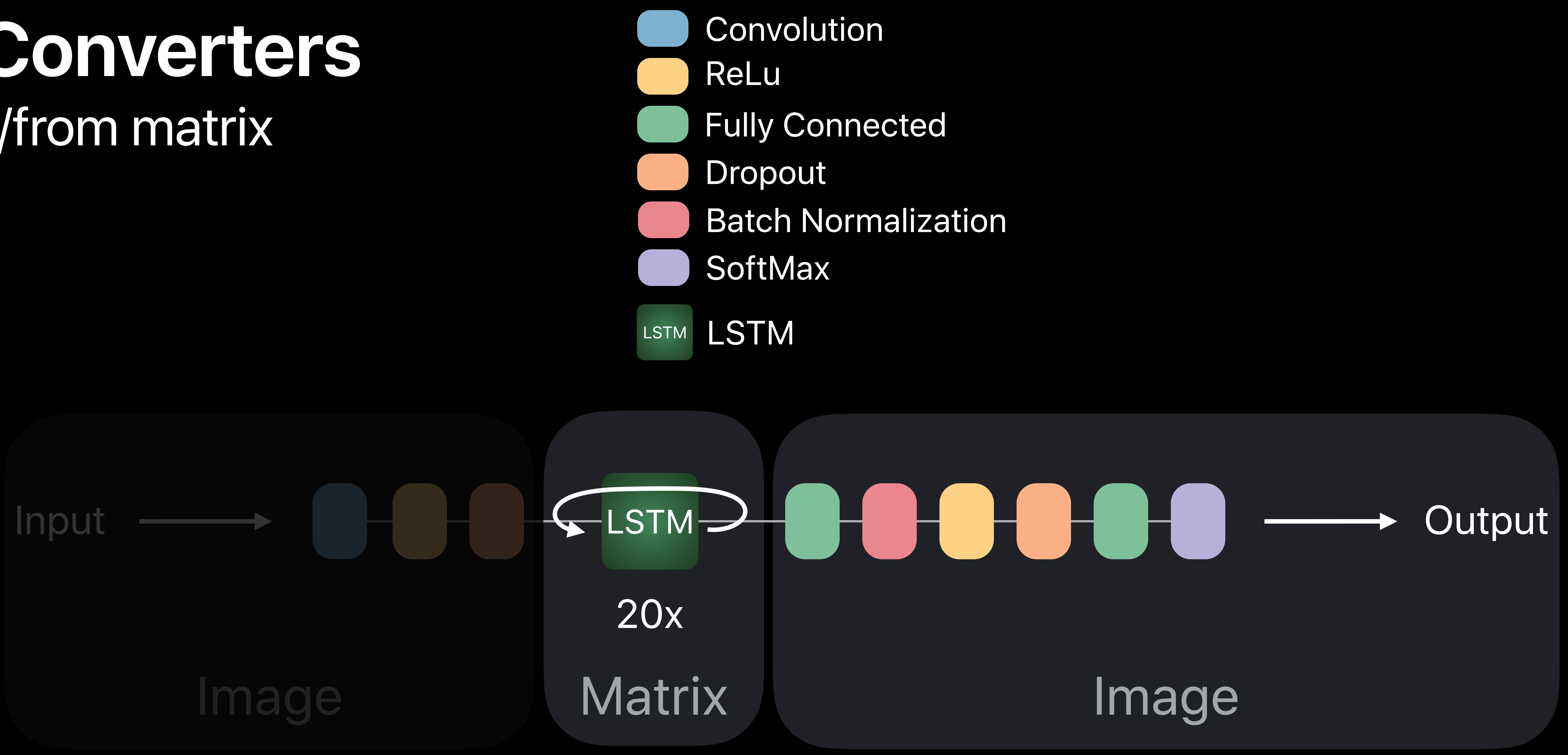
Image to/from matrix




```
let imageToMatrix = MPSImageCopyToMatrix(device: device, dataLayout: .featureChannelsxHeightxWidth)
imageToMatrix.encodeBatch(commandBuffer: commandBuffer, sourceImages: [image1, image2],
                          destinationMatrix: matrix)
```

Data Converters

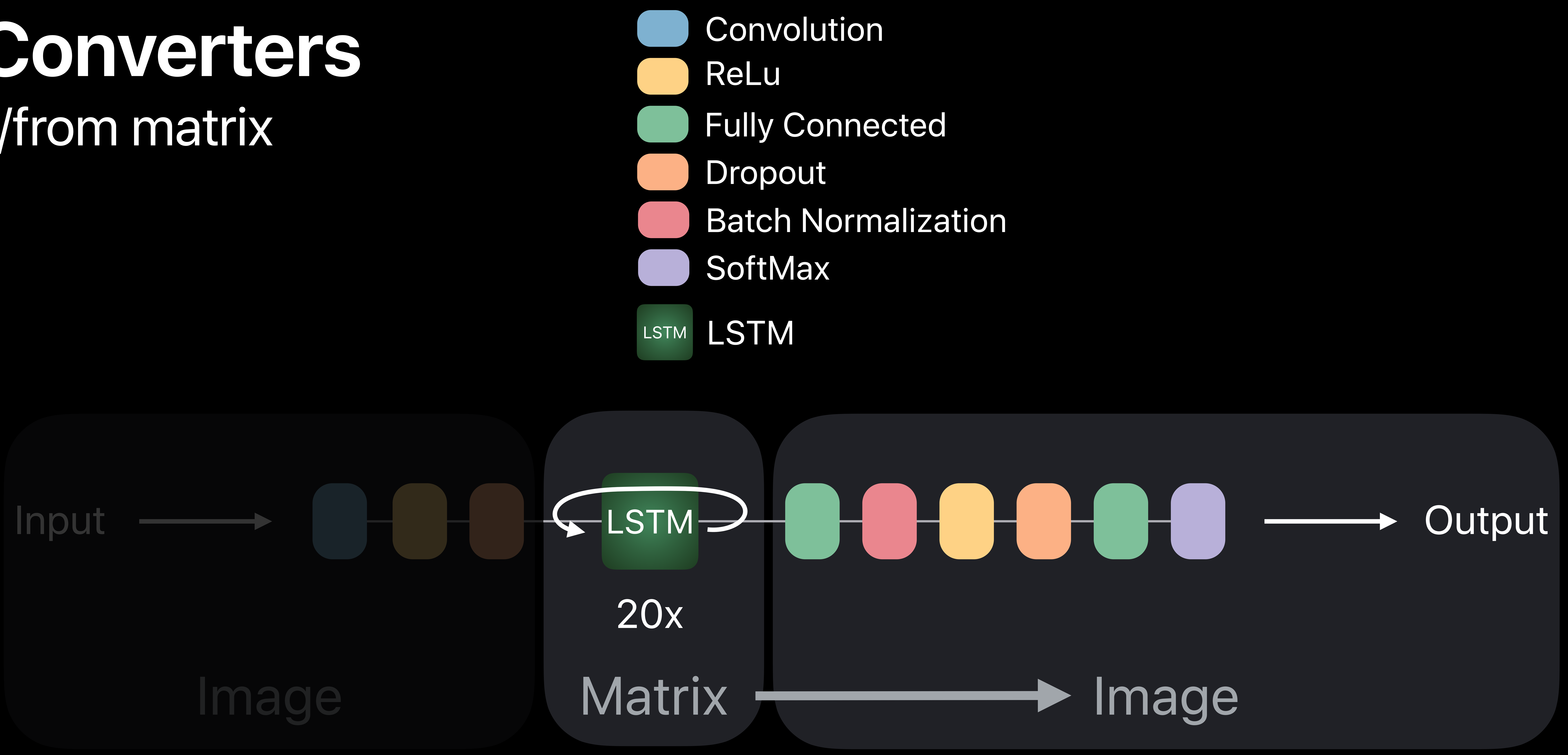
Image to/from matrix




```
let matrixToImage = MPSMatrixCopyToImage(device: device, dataLayout: .featureChannelsxHeightxWidth)
matrixToImage.encodeBatch(commandBuffer: commandBuffer, sourceMatrix: matrix,
                          destinationImages: [image1, image2])
```

Data Converters

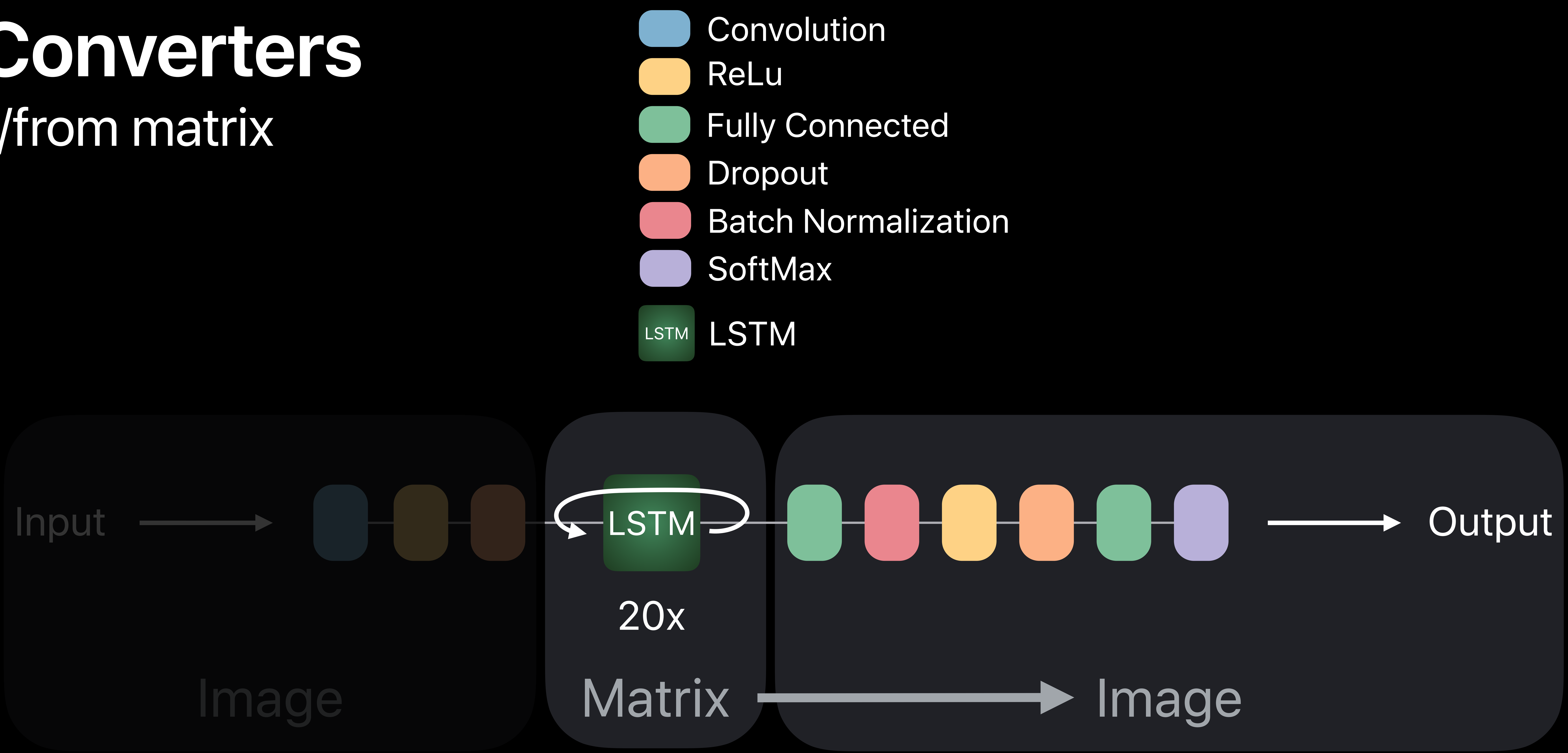
Image to/from matrix




```swift
let matrixToImage = MPSMatrixCopyToImage(device: device, dataLayout: .featureChannelsxHeightxWidth)
matrixToImage.encodeBatch(commandBuffer: commandBuffer, sourceMatrix: matrix,
                          destinationImages: [image1, image2])
```

Data Converters

Image to/from matrix




```
let matrixToImage = MPSMatrixCopyToImage(device: device, dataLayout: .featureChannelsxHeightxWidth)
matrixToImage.encodeBatch(commandBuffer: commandBuffer, sourceMatrix: matrix,
                          destinationImages: [image1, image2])
```

Demo

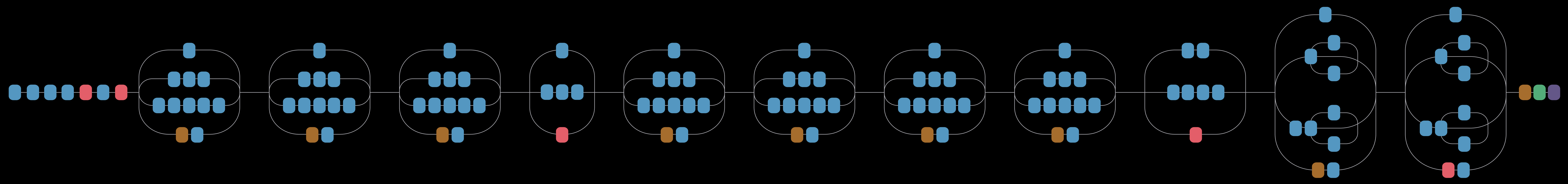
TensorFlow

Demo

Object classification training using TensorFlow with MPS

Demo

Object classification training using TensorFlow with MPS



InceptionV3 network*

*Rethinking the Inception Architecture for Computer Vision, Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna, CVPR 2015, https://arxiv.org/abs/1512.00567

Summary

FP16 accumulation for inference

GPU-accelerated primitives

- For training neural networks
- Optimized for iOS and macOS
- Neural Network Graph API for ease of use

More Information

https://developer.apple.com/wwdc18/609

Metal for Machine Learning Lab	Technology Lab 5	Friday 9:00AM

WWDC18