#WWDC19

Metal for Machine Learning

Justin Voo, GPU Software Engineer

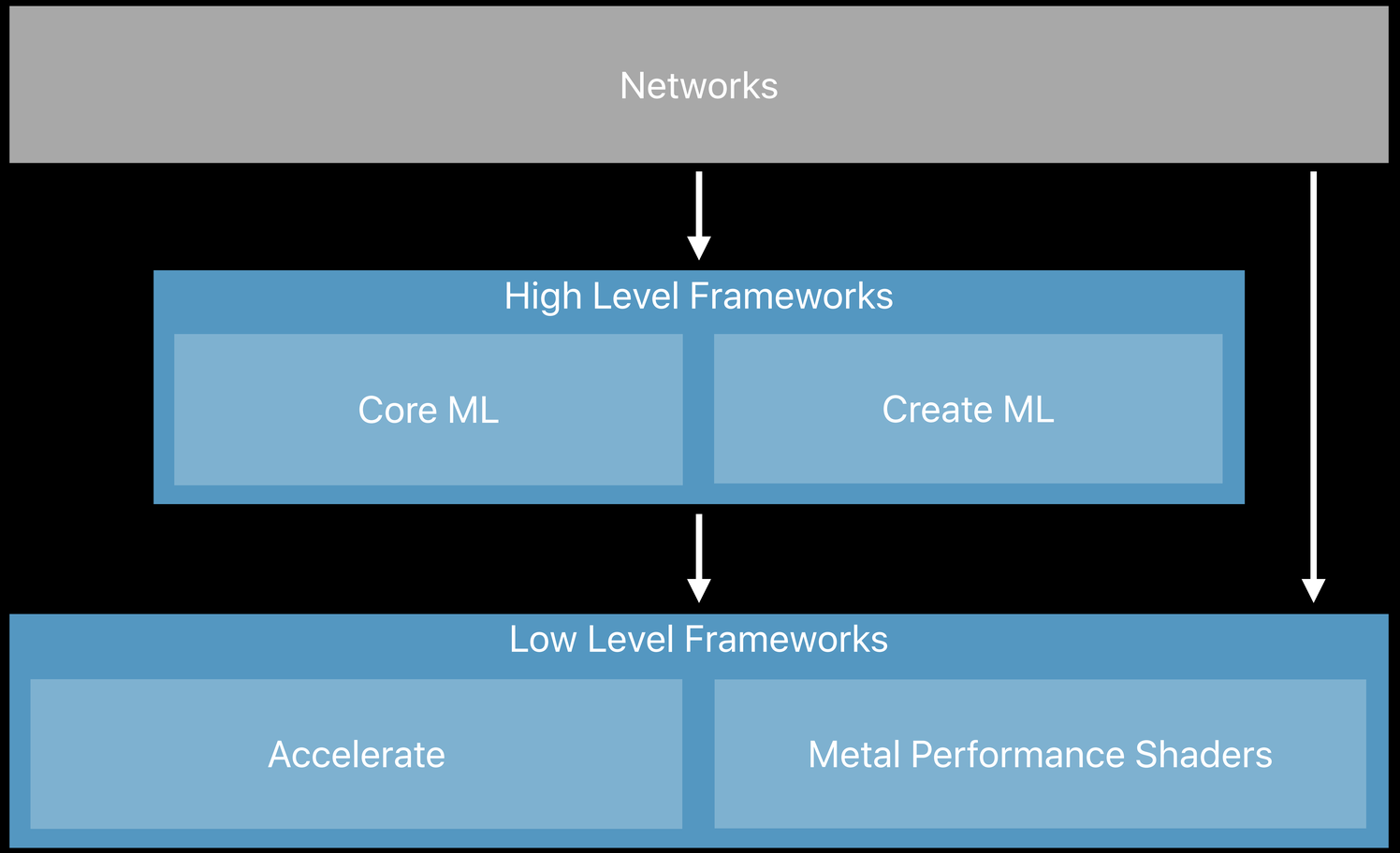
Metal Performance Shaders

GPU-accelerated primitives

- Image processing
- Linear algebra
- Ray tracing
- Machine learning

Optimized for iOS, macOS, and tvOS

Metal Performance Shaders



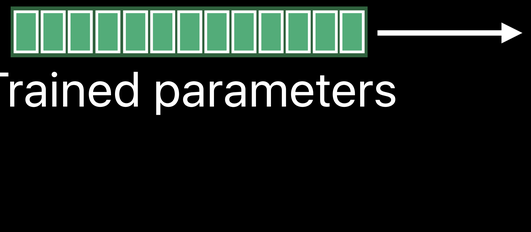
Metal for Machine Learning

NEW

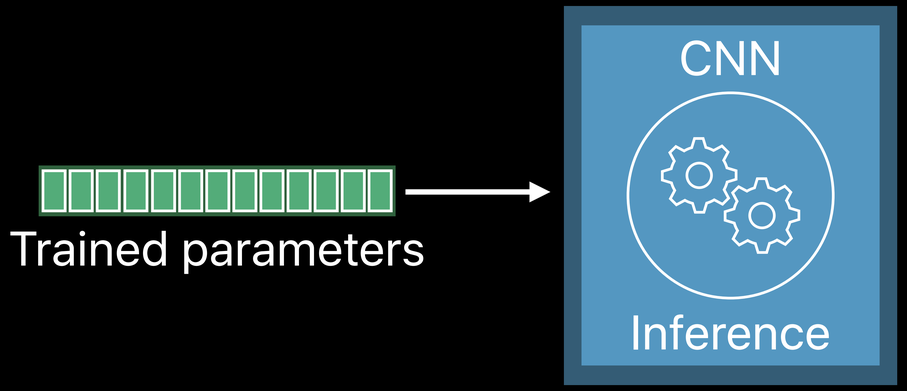
What’s new in MPS this year

- Support for new networks
- Improved performance
- Easier to use

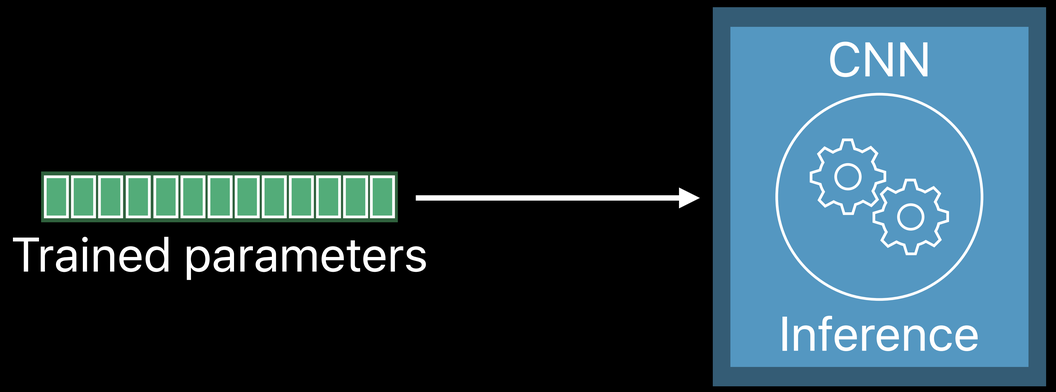
Inference



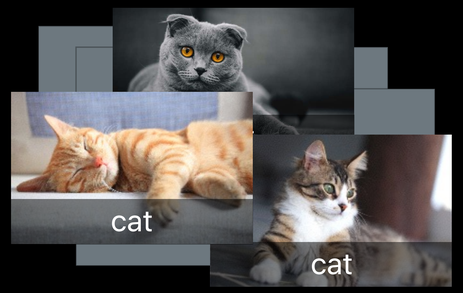
Inference

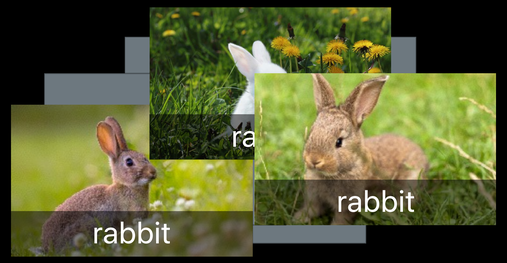


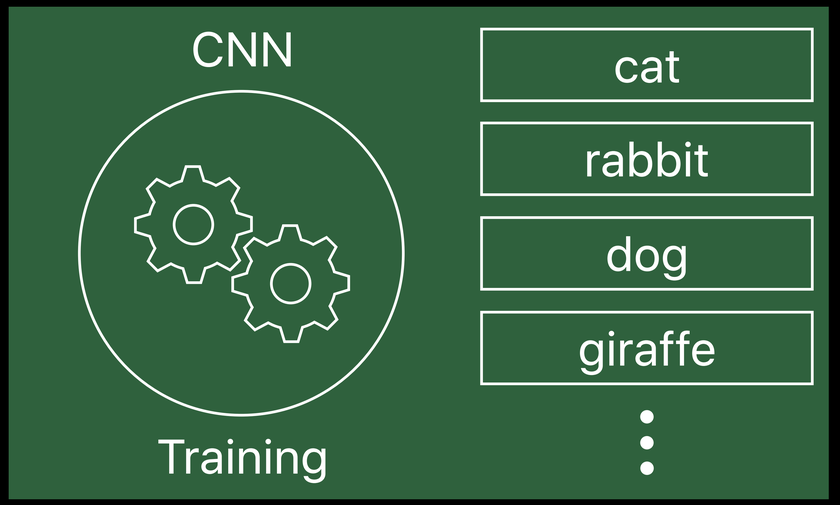
Inference



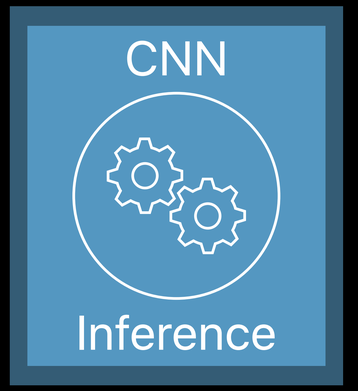
Training



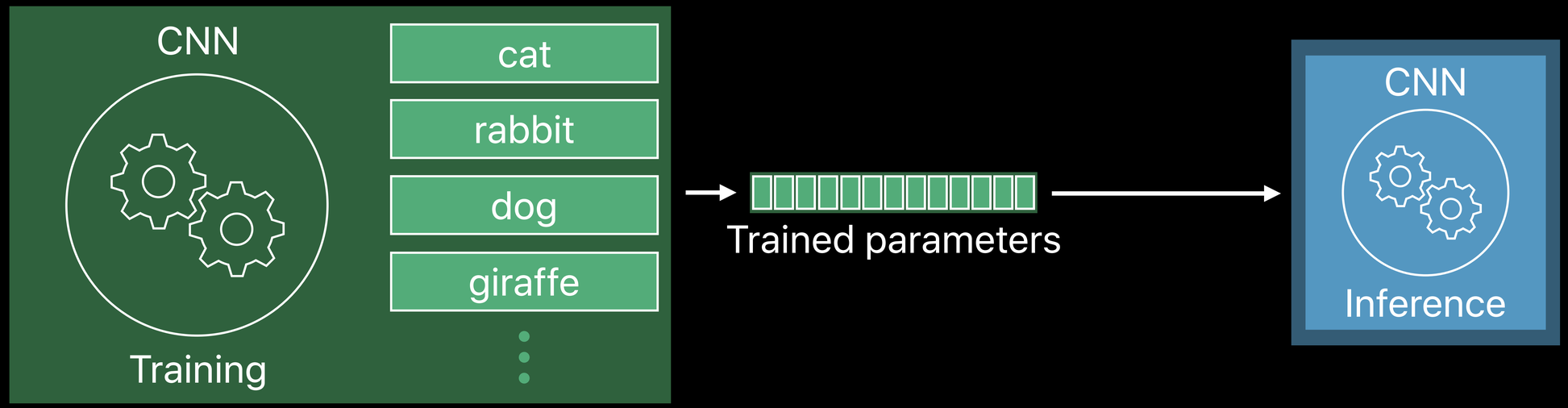




Trained parameters



Training



Using Metal 2 for Compute	WWDC 2017
Metal for Accelerating Machine Learning	WWDC 2018

New Features

NEW

Implicit graph

Separable loss

Random number generation

Predication

Commit and continue

New Features

Implicit graph

Separable loss

Random number generation

Predication

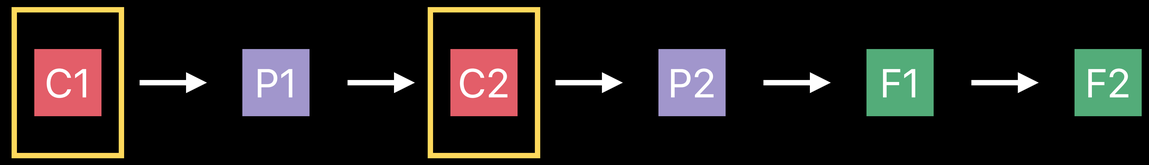
Commit and continue

Inference Graph

Review

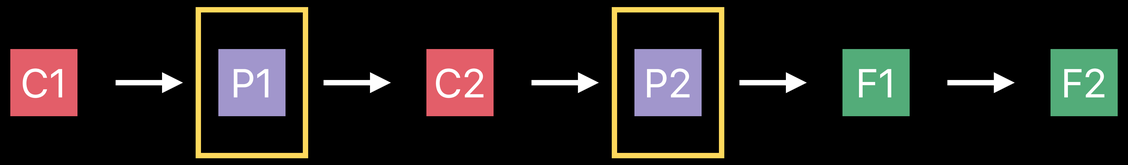
Inference Graph

Review



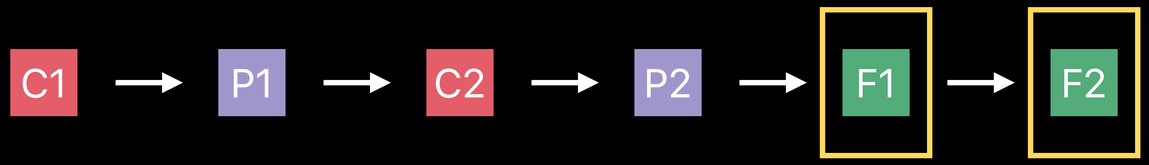
Inference Graph

Review



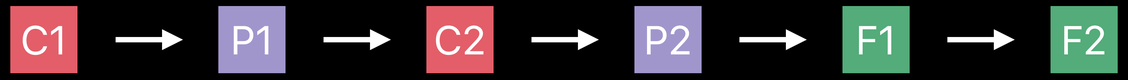
Inference Graph

Review



Inference Graph

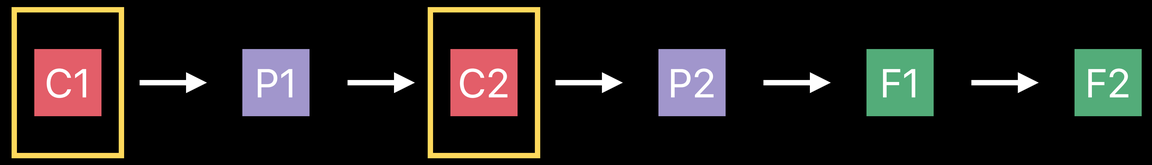
Review




```
let C1 = MPSCNNConvolutionNode(source: MPSNNImageNode(handle: nil),
                               weights: MyWeights(file:“conv1.dat”))
let P1 = MPSCNNPoolingMaxNode(source: C1.resultImage, filterSize: 2)
let C2 = MPSCNNConvolutionNode(source: P1.resultImage, weights: MyWeights(file:“conv2.dat”))
let P2 = MPSCNNPoolingMaxNode(source: C2.resultImage, filterSize: 2)
let F1 = MPSCNNFullyConnectedNode(source: P2.resultImage, weights: MyWeights(file:“fc1.dat”))
let F2 = MPSCNNFullyConnectedNode(source: F1.resultImage, weights: MyWeights(file:“fc2.dat”))
```

Inference Graph

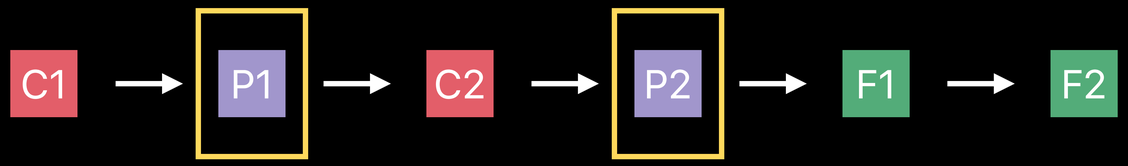
Review




```
let C1 = MPSCNNConvolutionNode(source: MPSNNImageNode(handle: nil),
                               weights: MyWeights(file:"conv1.dat"))
let P1 = MPSCNNPoolingMaxNode(source: C1.resultImage, filterSize: 2)
let C2 = MPSCNNConvolutionNode(source: P1.resultImage, weights: MyWeights(file:"conv2.dat"))
let P2 = MPSCNNPoolingMaxNode(source: C2.resultImage, filterSize: 2)
let F1 = MPSCNNFullyConnectedNode(source: P2.resultImage, weights: MyWeights(file:"fc1.dat"))
let F2 = MPSCNNFullyConnectedNode(source: F1.resultImage, weights: MyWeights(file:"fc2.dat"))
```

Inference Graph

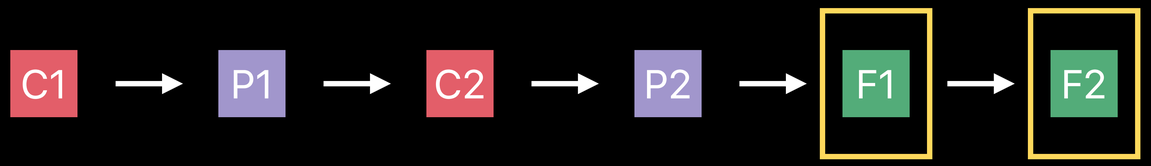
Review




```
let C1 = MPSCNNConvolutionNode(source: MPSNNImageNode(handle: nil),
                               weights: MyWeights(file:“conv1.dat”))
let P1 = MPSCNNPoolingMaxNode(source: C1.resultImage, filterSize: 2)
let C2 = MPSCNNConvolutionNode(source: P1.resultImage, weights: MyWeights(file:“conv2.dat”))
let P2 = MPSCNNPoolingMaxNode(source: C2.resultImage, filterSize: 2)
let F1 = MPSCNNFullyConnectedNode(source: P2.resultImage, weights: MyWeights(file:“fc1.dat”))
let F2 = MPSCNNFullyConnectedNode(source: F1.resultImage, weights: MyWeights(file:“fc2.dat”))
```

Inference Graph

Review



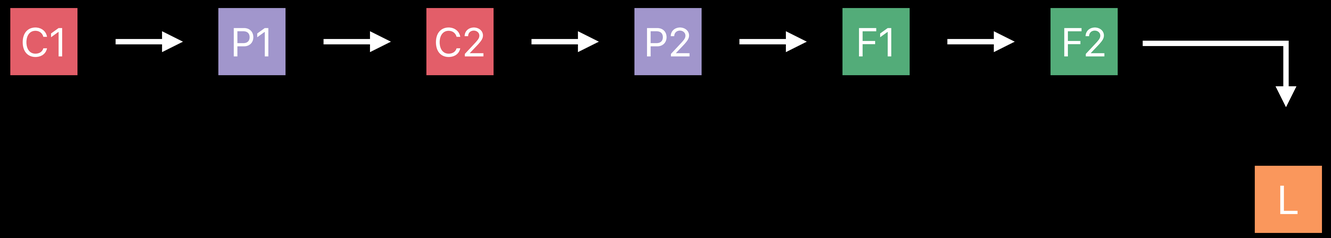

```
let C1 = MPSCNNConvolutionNode(source: MPSNNImageNode(handle: nil),
                               weights: MyWeights(file:“conv1.dat”))
let P1 = MPSCNNPoolingMaxNode(source: C1.resultImage, filterSize: 2)
let C2 = MPSCNNConvolutionNode(source: P1.resultImage, weights: MyWeights(file:“conv2.dat”))
let P2 = MPSCNNPoolingMaxNode(source: C2.resultImage, filterSize: 2)
let F1 = MPSCNNFullyConnectedNode(source: P2.resultImage, weights: MyWeights(file:“fc1.dat”))
let F2 = MPSCNNFullyConnectedNode(source: F1.resultImage, weights: MyWeights(file:“fc2.dat”))
```

Training Graph

Review

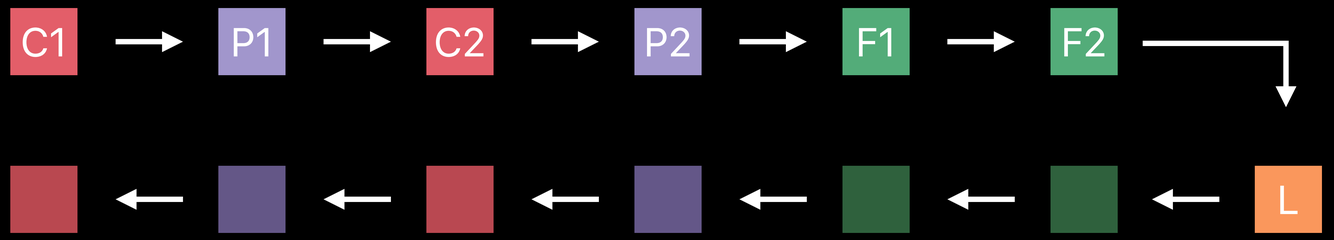
Training Graph

Review



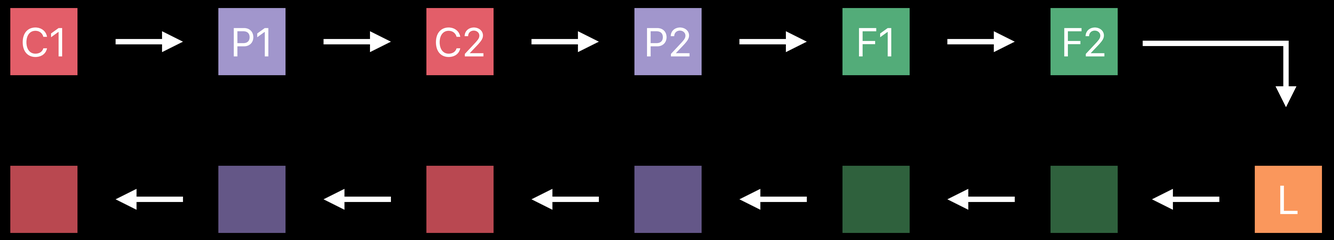
Training Graph

Review



Training Graph

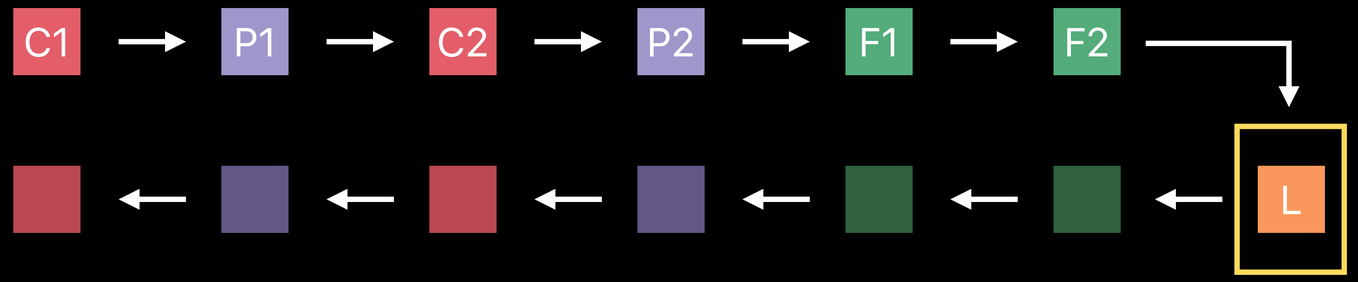
Review




```
let L   = MPSCNNLossNode(source: F2.resultImage, lossDescriptor: lossDescriptor)
let F1G = F1.gradientFilter(withSource: L.resultImage)
let F2G = F2.gradientFilter(withSource: F1G.resultImage)
let P2G = P2.gradientFilter(withSource: F2G.resultImage)
let C2G = C2.gradientFilter(withSource: P2G.resultImage)
let P1G = P1.gradientFilter(withSource: C2G.resultImage)
let C1G = C1.gradientFilter(withSource: P1G.resultImage)
```

Training Graph

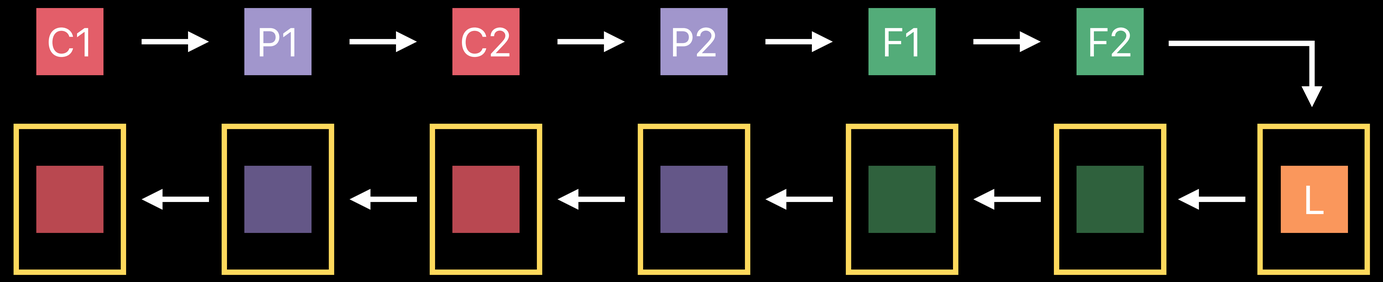
Review




```
let L   = MPSCNNLossNode(source: F2.resultImage, lossDescriptor: lossDescriptor)
let F1G = F1.gradientFilter(withSource: L.resultImage)
let F2G = F2.gradientFilter(withSource: F1G.resultImage)
let P2G = P2.gradientFilter(withSource: F2G.resultImage)
let C2G = C2.gradientFilter(withSource: P2G.resultImage)
let P1G = P1.gradientFilter(withSource: C2G.resultImage)
let C1G = C1.gradientFilter(withSource: P1G.resultImage)
```

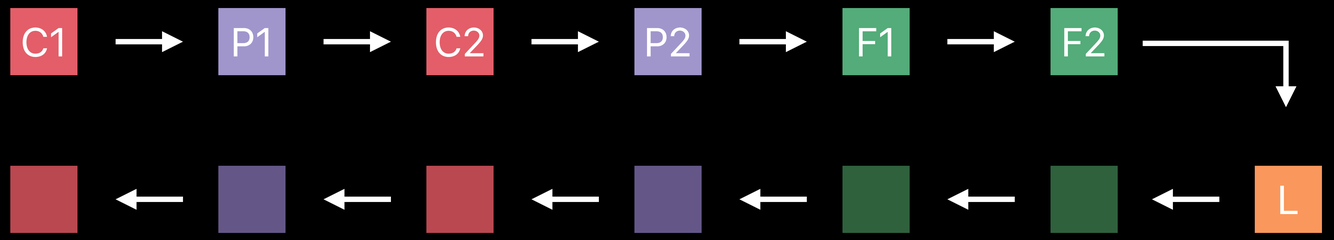
Training Graph

Review



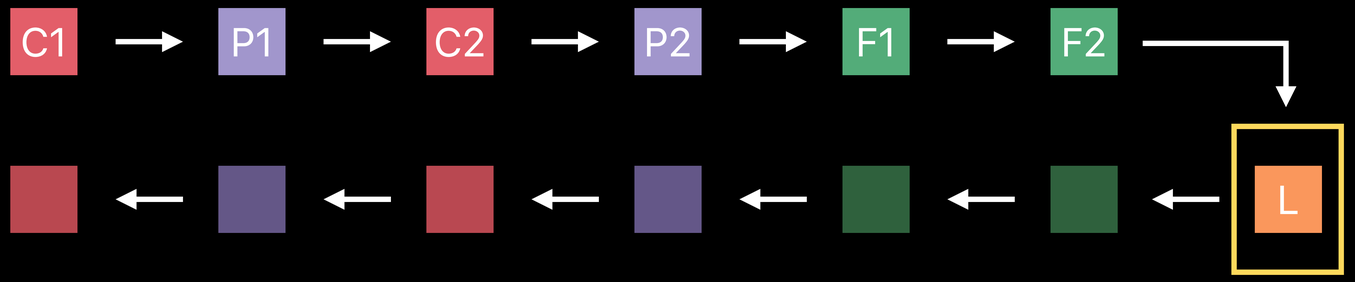

```
let L   = MPSCNNLossNode(source: F2.resultImage, lossDescriptor: lossDescriptor)
let F1G = F1.gradientFilter(withSource: L.resultImage)
let F2G = F2.gradientFilter(withSource: F1G.resultImage)
let P2G = P2.gradientFilter(withSource: F2G.resultImage)
let C2G = C2.gradientFilter(withSource: P2G.resultImage)
let P1G = P1.gradientFilter(withSource: C2G.resultImage)
let C1G = C1.gradientFilter(withSource: P1G.resultImage)
```

Implicit Graph



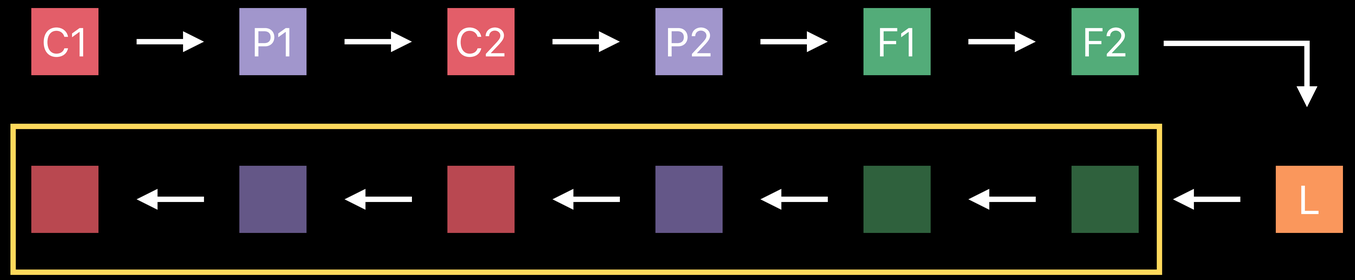

```
let L   = MPSCNNLossNode(source: F2.resultImage, lossDescriptor: lossDescriptor)
let gradientNodes = L.trainingGraph(withSourceGradient: nil, nodeHandler: nil)
```

Implicit Graph




```
let L   = MPSCNNLossNode(source: F2.resultImage, lossDescriptor: lossDescriptor)
let gradientNodes = L.trainingGraph(withSourceGradient: nil, nodeHandler: nil)
```

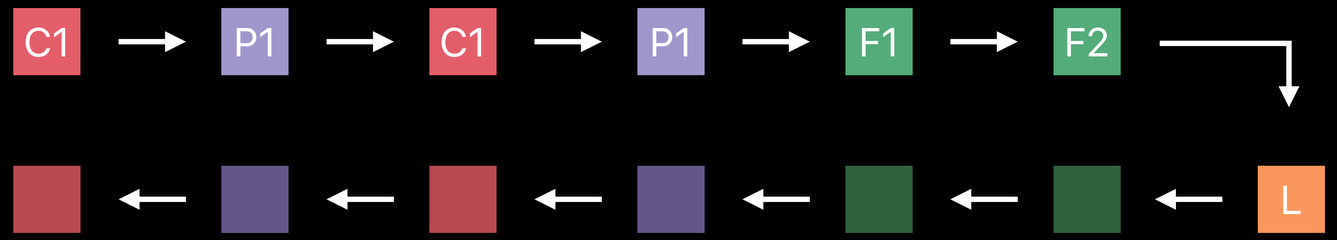
Implicit Graph




```
let L   = MPSCNNLossNode(source: F2.resultImage, lossDescriptor: lossDescriptor)
let gradientNodes = L.trainingGraph(withSourceGradient: nil, nodeHandler: nil)
```

Implicit Graph

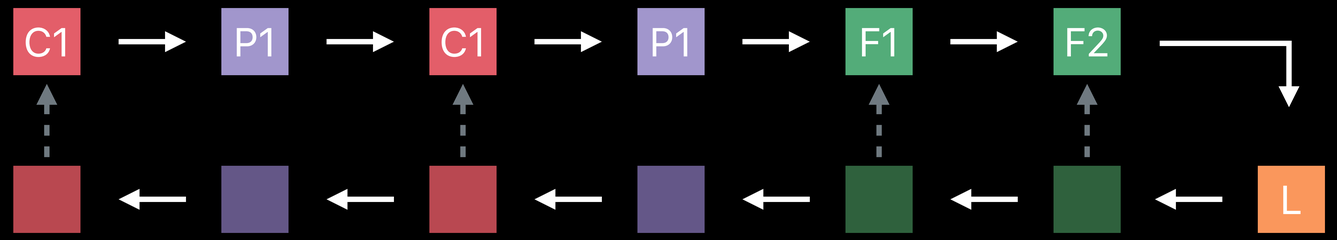
Stop gradient




```
let L  = MPSCNNLossNode(source: F2.resultImage, lossDescriptor: lossDescriptor)
F1.resultImage.stopGradient = true
let gradientNodes = L.trainingGraph(withSourceGradient: nil, nodeHandler: nil)
```

Implicit Graph

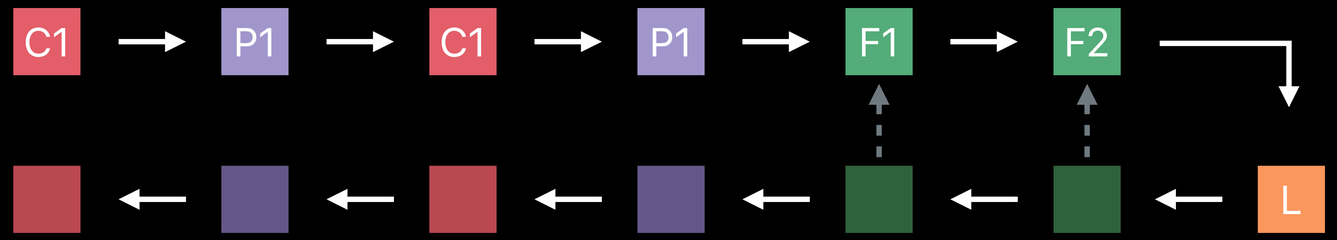
Stop gradient




```
let L   = MPSCNNLossNode(source: F2.resultImage, lossDescriptor: lossDescriptor)
F1.resultImage.stopGradient = true
let gradientNodes = L.trainingGraph(withSourceGradient: nil, nodeHandler: nil)
```

Implicit Graph

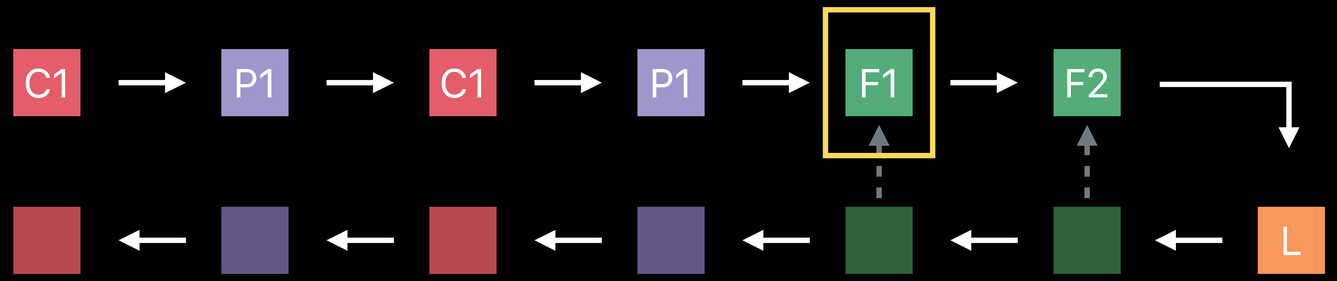
Stop gradient




```
let L   = MPSCNNLossNode(source: F2.resultImage, lossDescriptor: lossDescriptor)
F1.resultImage.stopGradient = true
let gradientNodes = L.trainingGraph(withSourceGradient: nil, nodeHandler: nil)
```

Implicit Graph

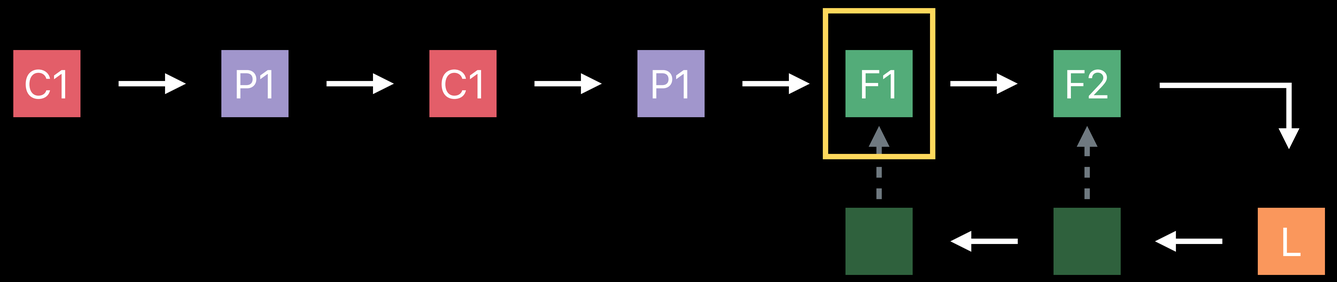
Stop gradient




```
let L   = MPSCNNLossNode(source: F2.resultImage, lossDescriptor: lossDescriptor)
F1.resultImage.stopGradient = true
let gradientNodes = L.trainingGraph(withSourceGradient: nil, nodeHandler: nil)
```

Implicit Graph

Stop gradient




```
let L   = MPSCNNLossNode(source: F2.resultImage, lossDescriptor: lossDescriptor)
F1.resultImage.stopGradient = true
let gradientNodes = L.trainingGraph(withSourceGradient: nil, nodeHandler: nil)
```

New Features

Implicit graph

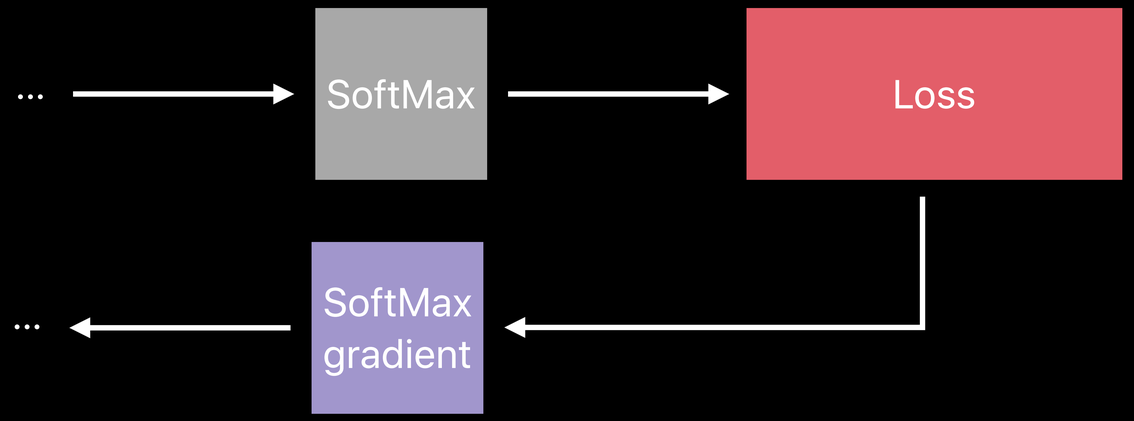
Separable loss

Random number generation

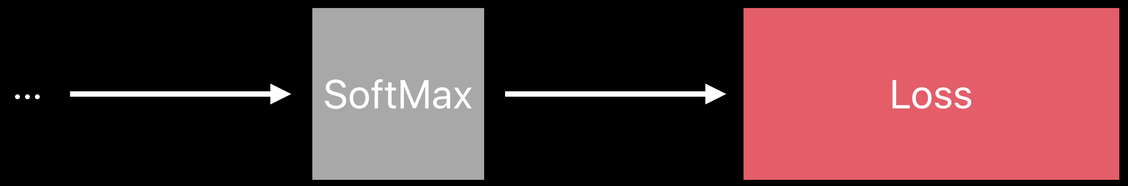
Predication

Commit and continue

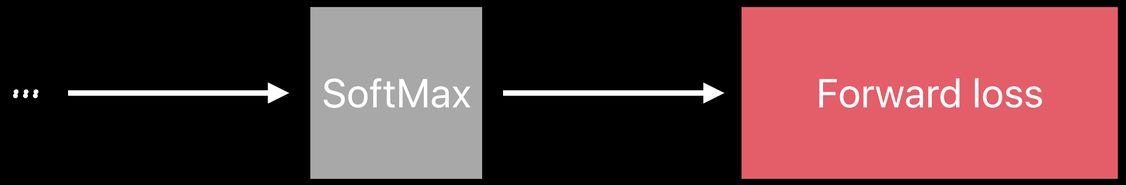
Separable Loss



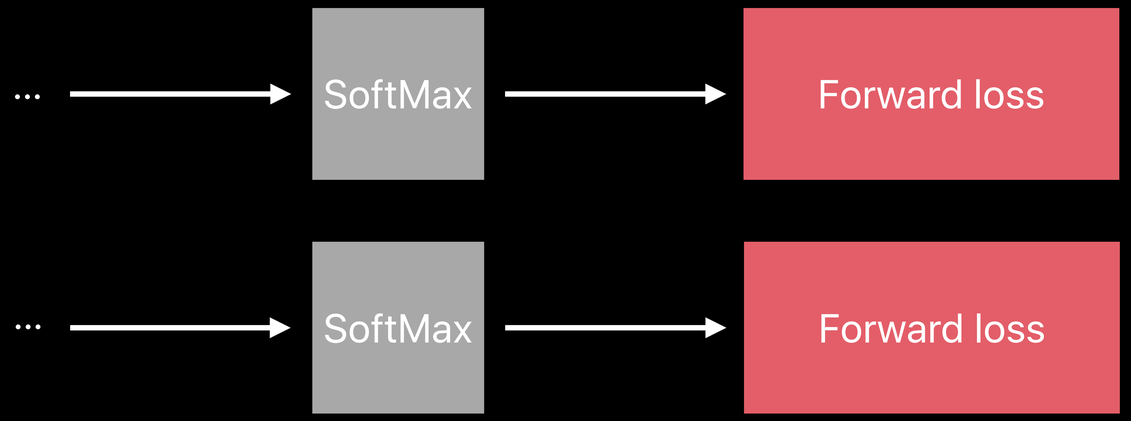
Separable Loss



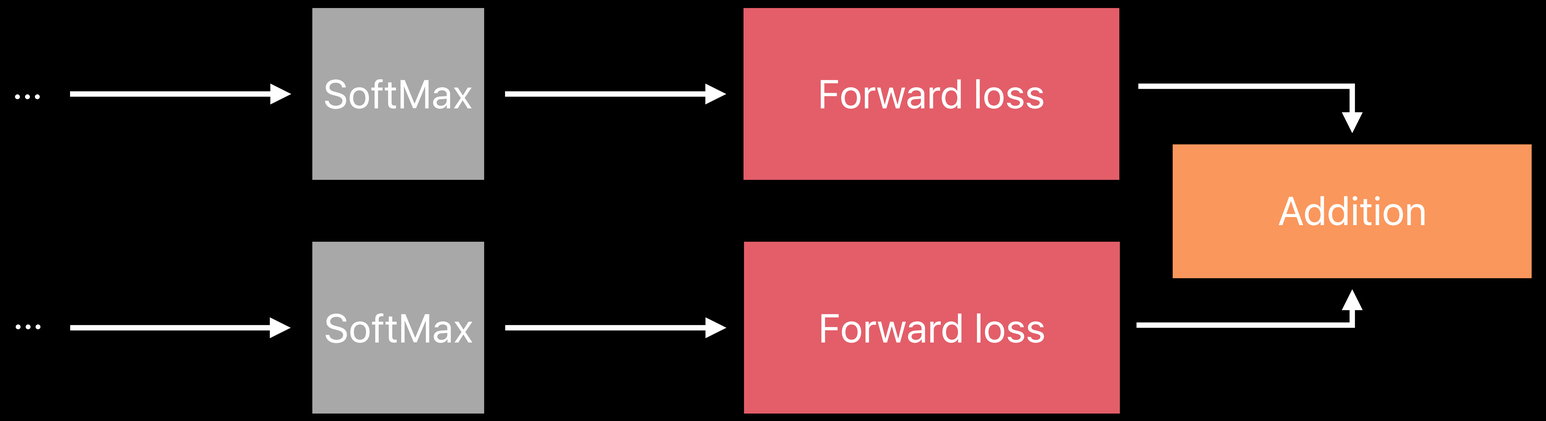
Separable Loss



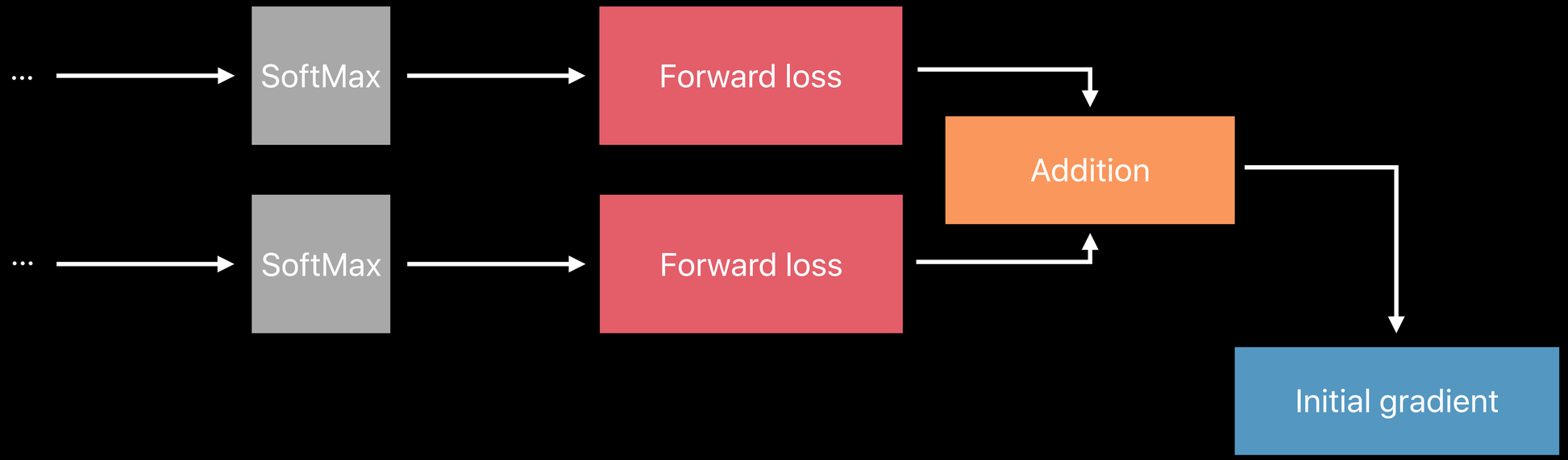
Separable Loss



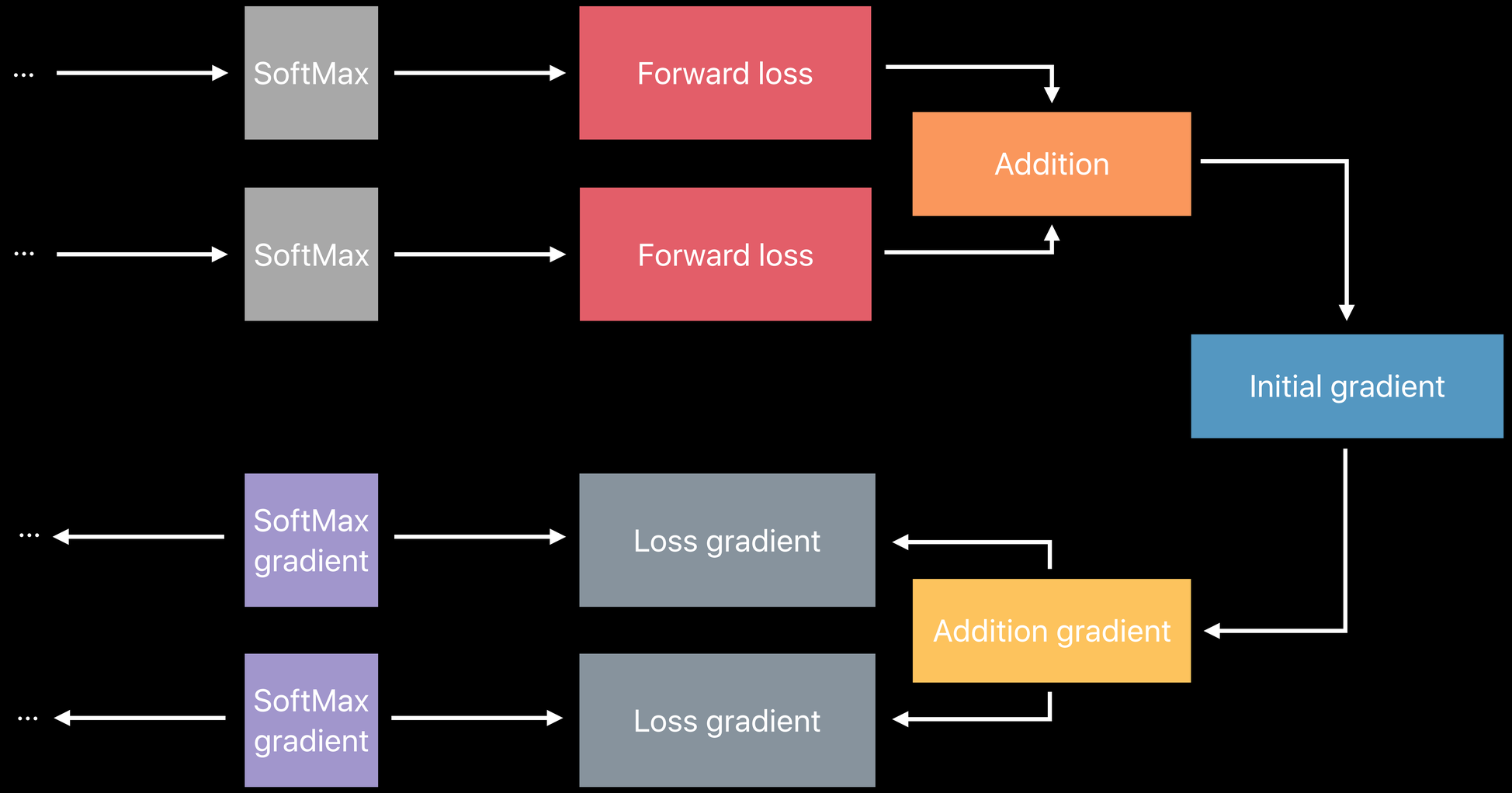
Separable Loss



Separable Loss

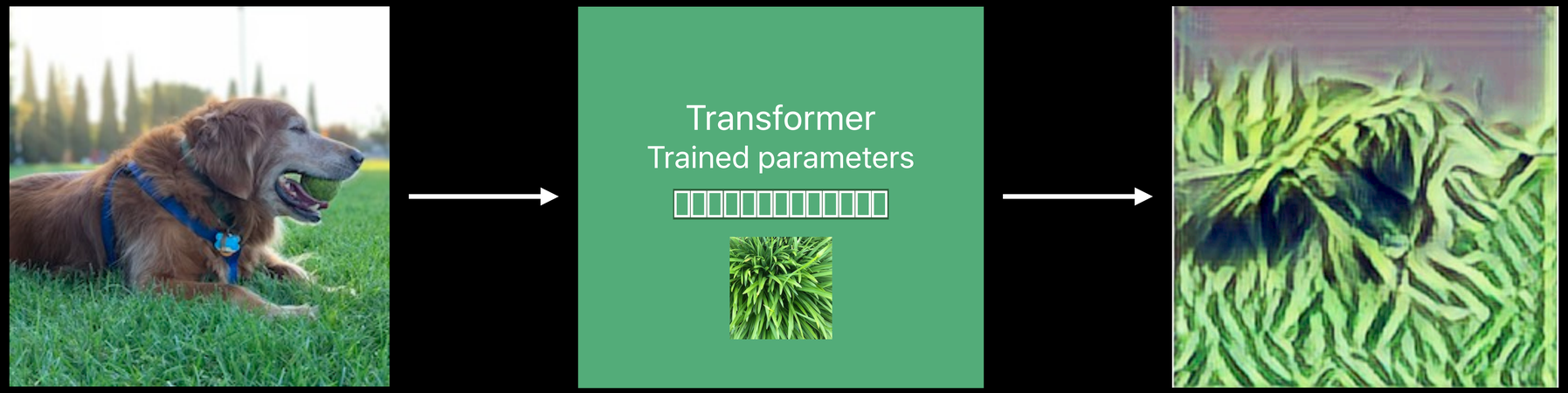


Separable Loss



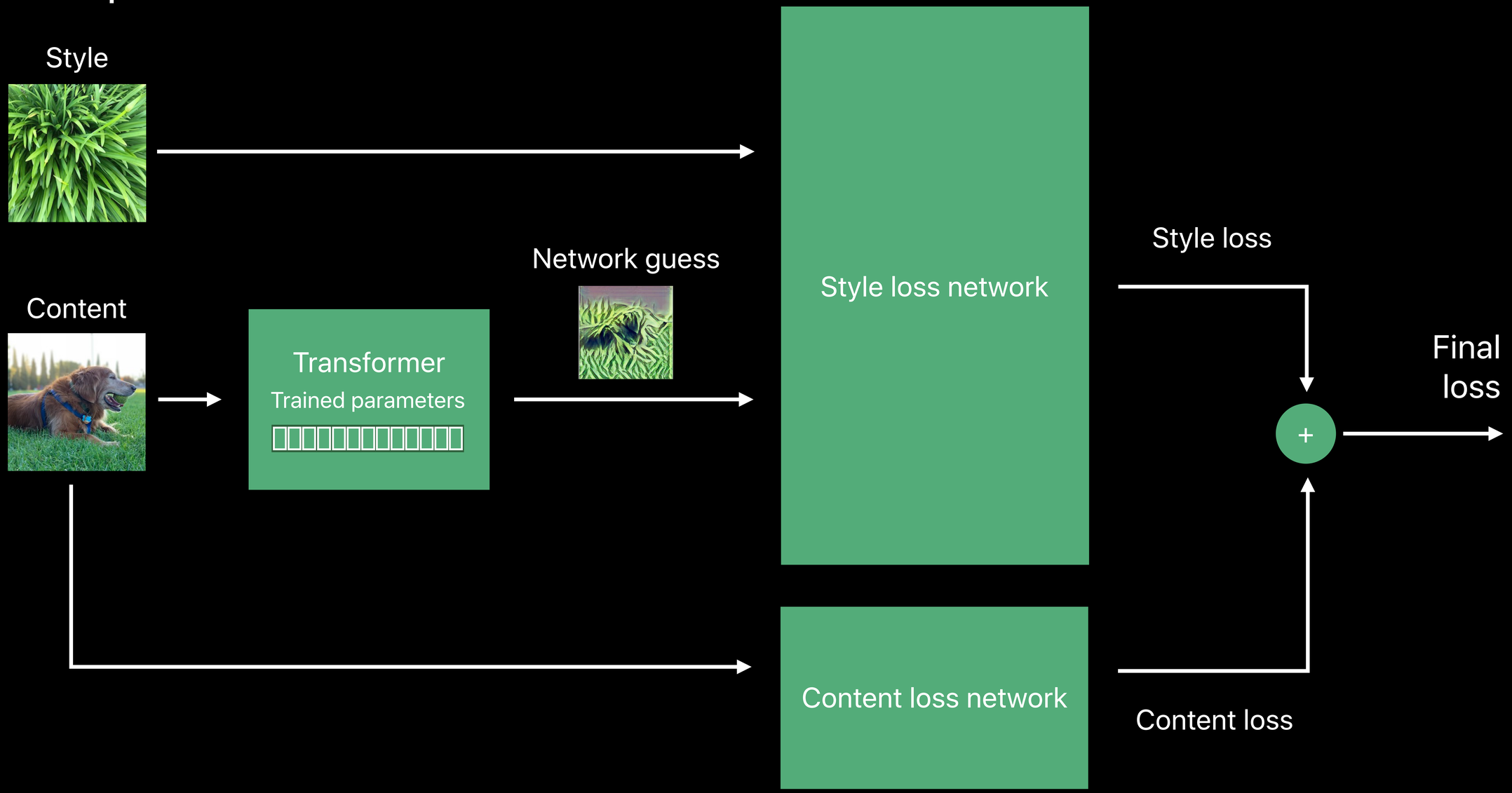
Style Transfer — Inference

Example



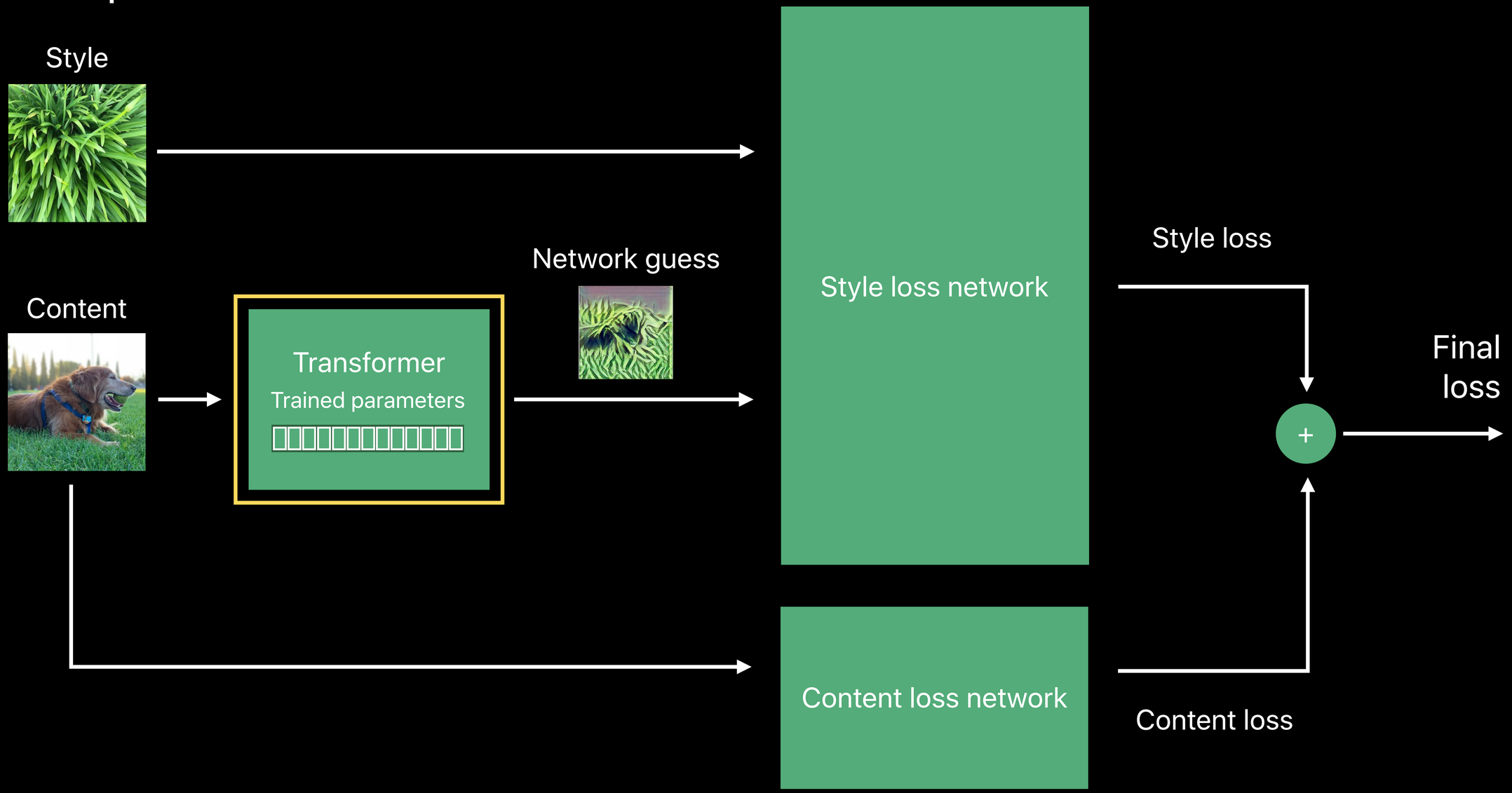
Style Transfer — Training

Example



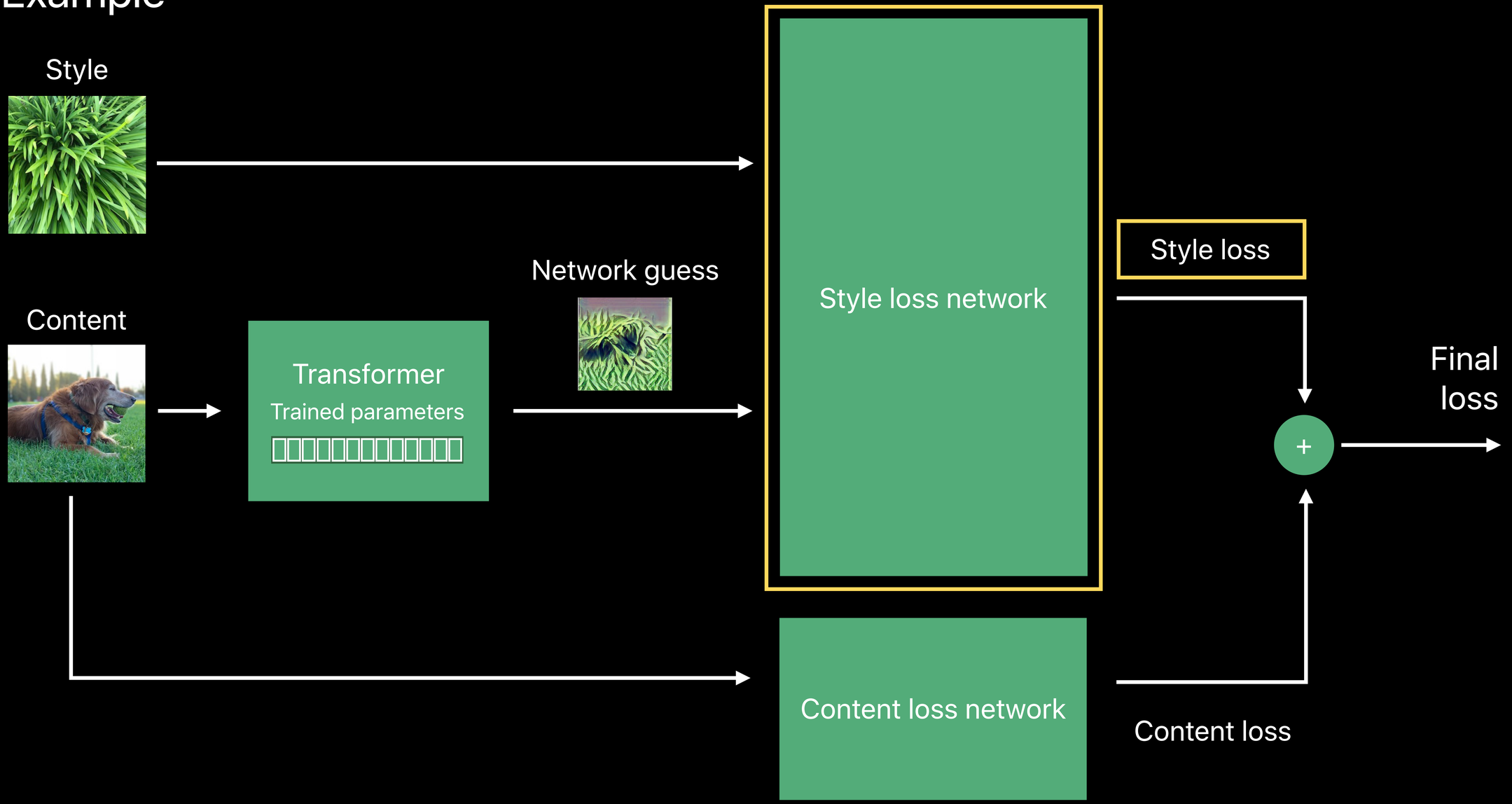
Style Transfer — Training

Example



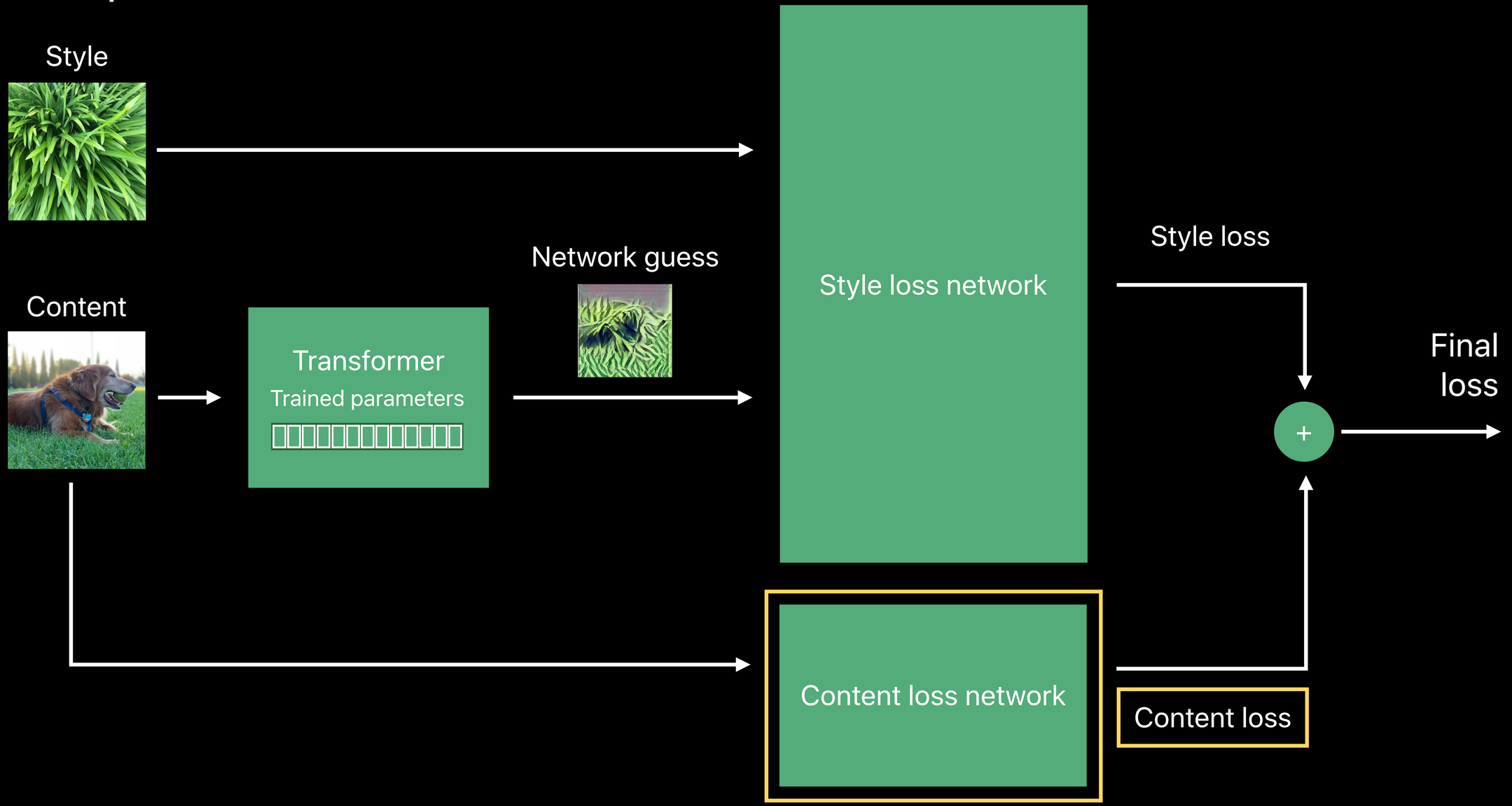
Style Transfer — Training

Example



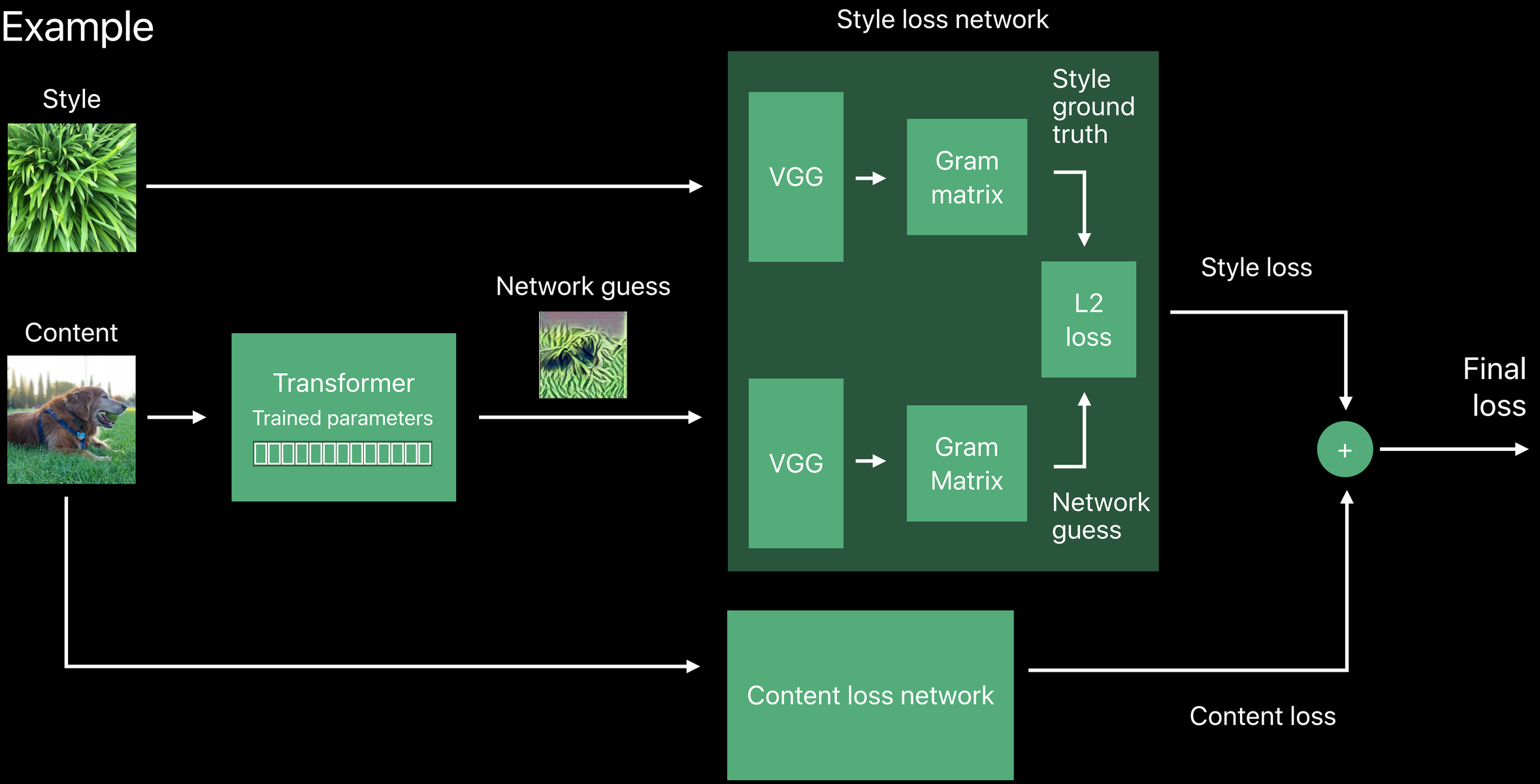
Style Transfer — Training

Example



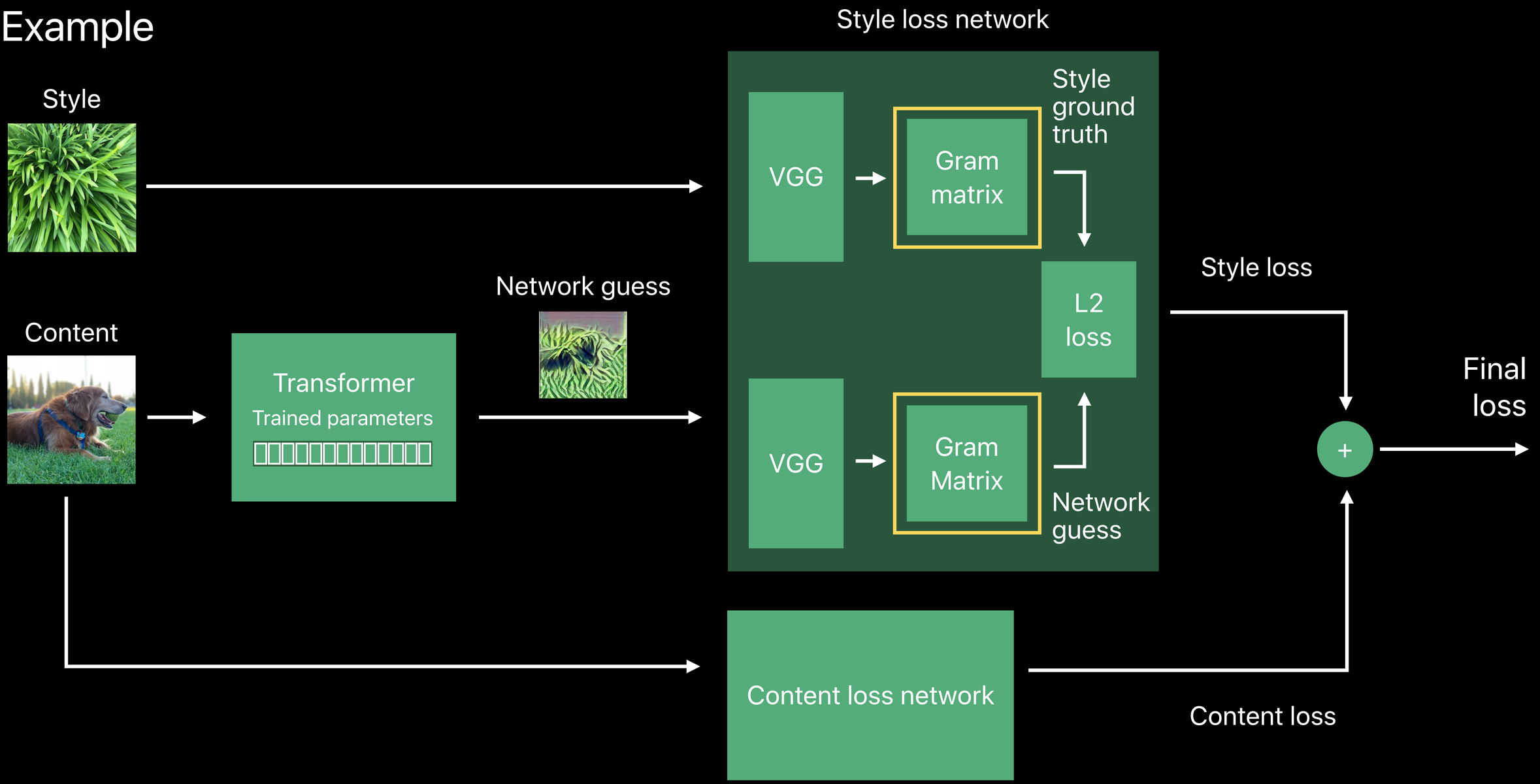
Style Transfer — Training

Example



Style Transfer — Training

Example

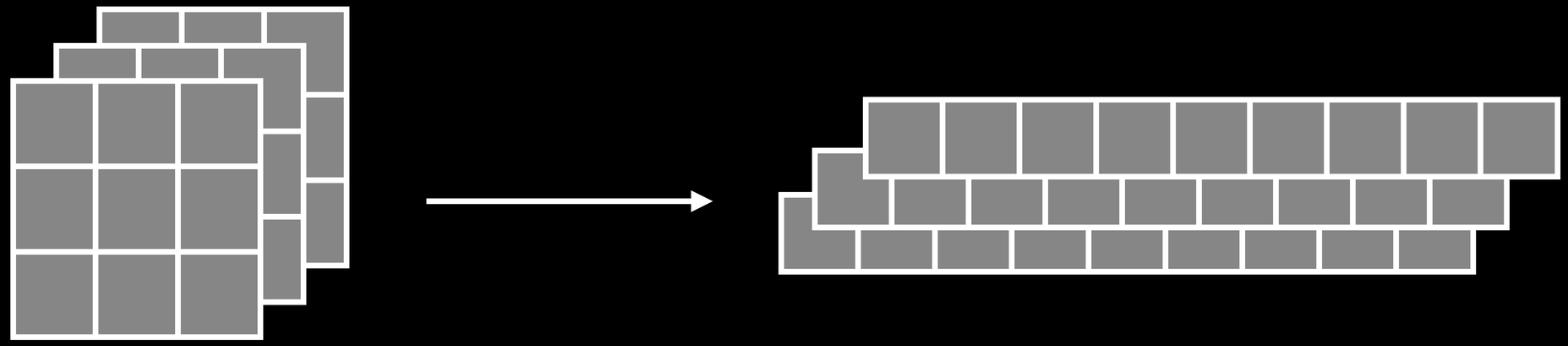


Gram Matrix

Calculate correlations between features in an image

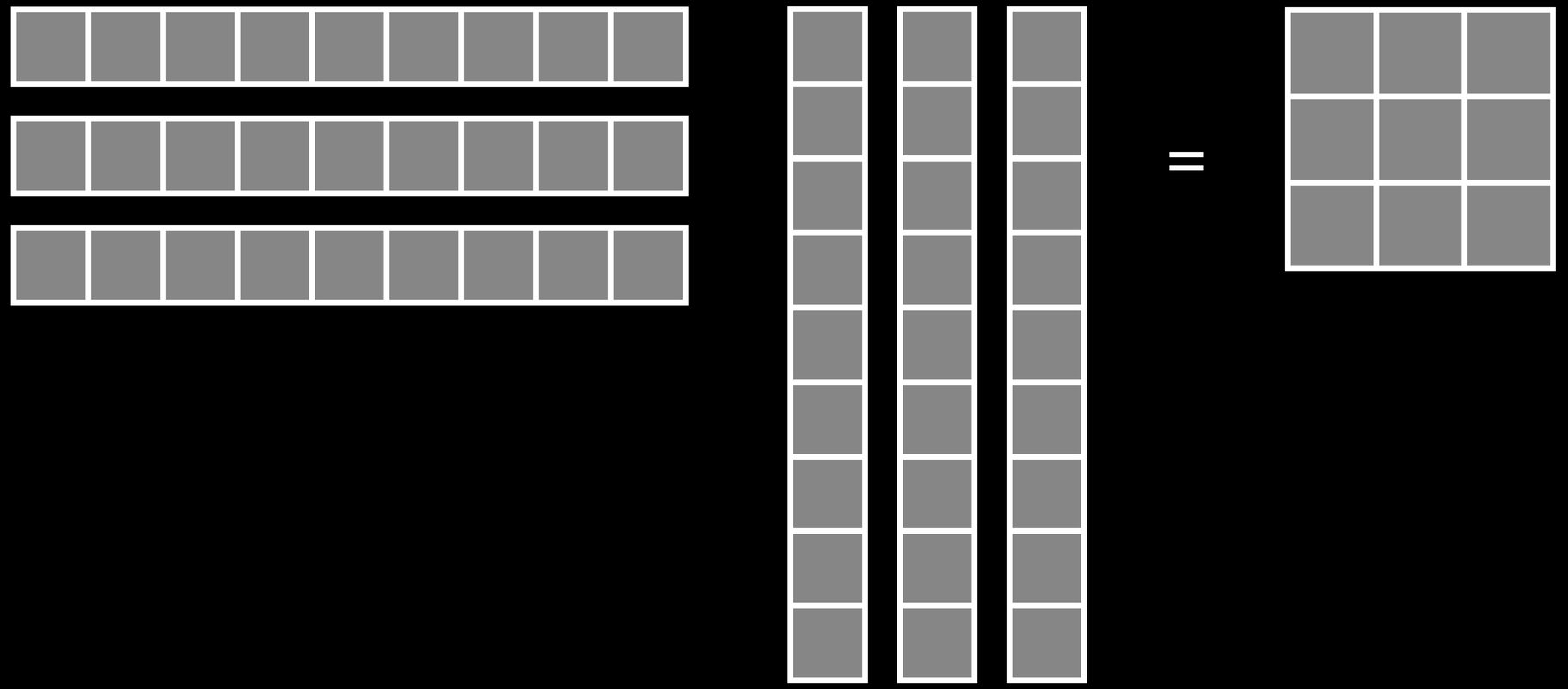
Gram Matrix

Calculate correlations between features in an image



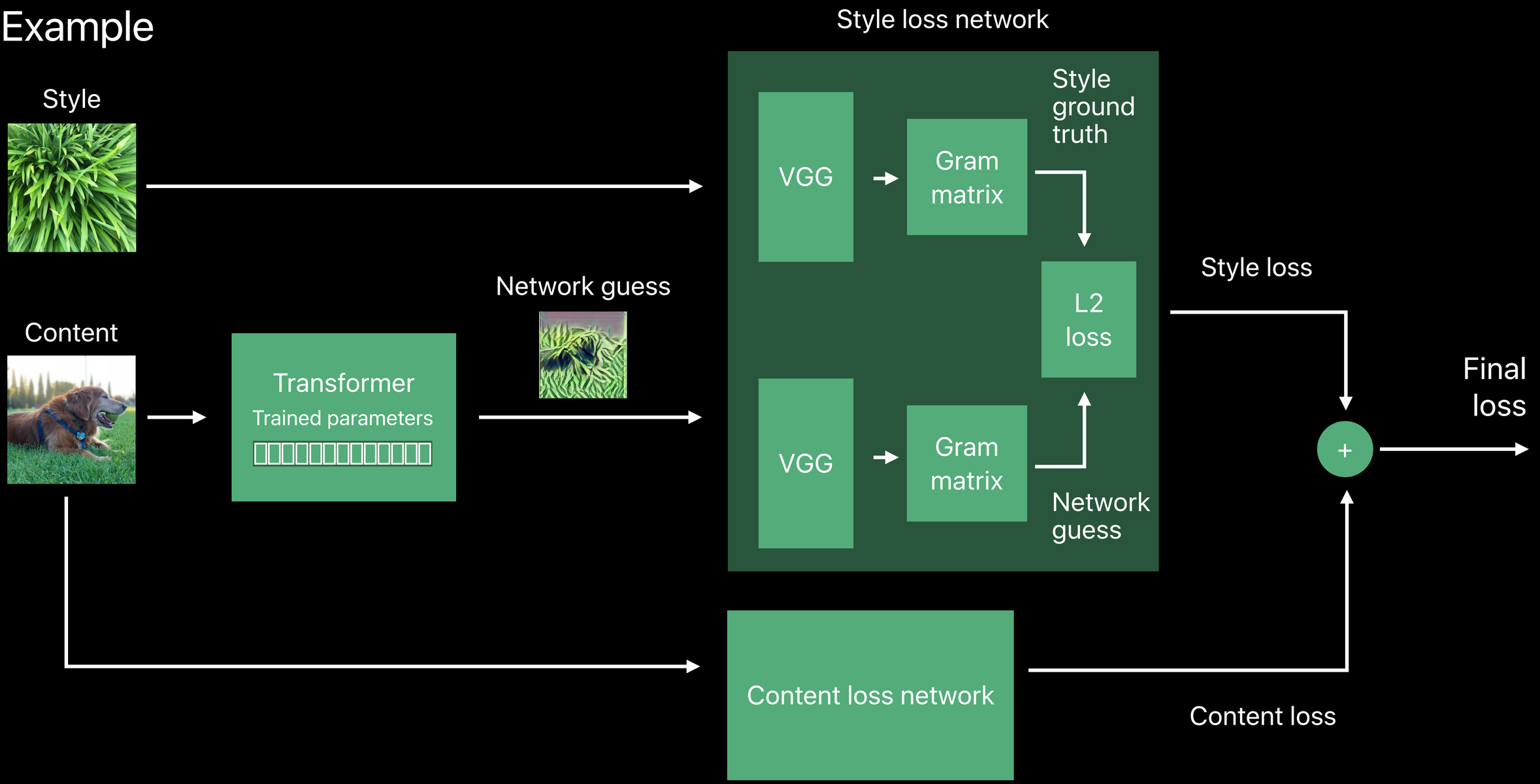
Gram Matrix

Calculate correlations between features in an image



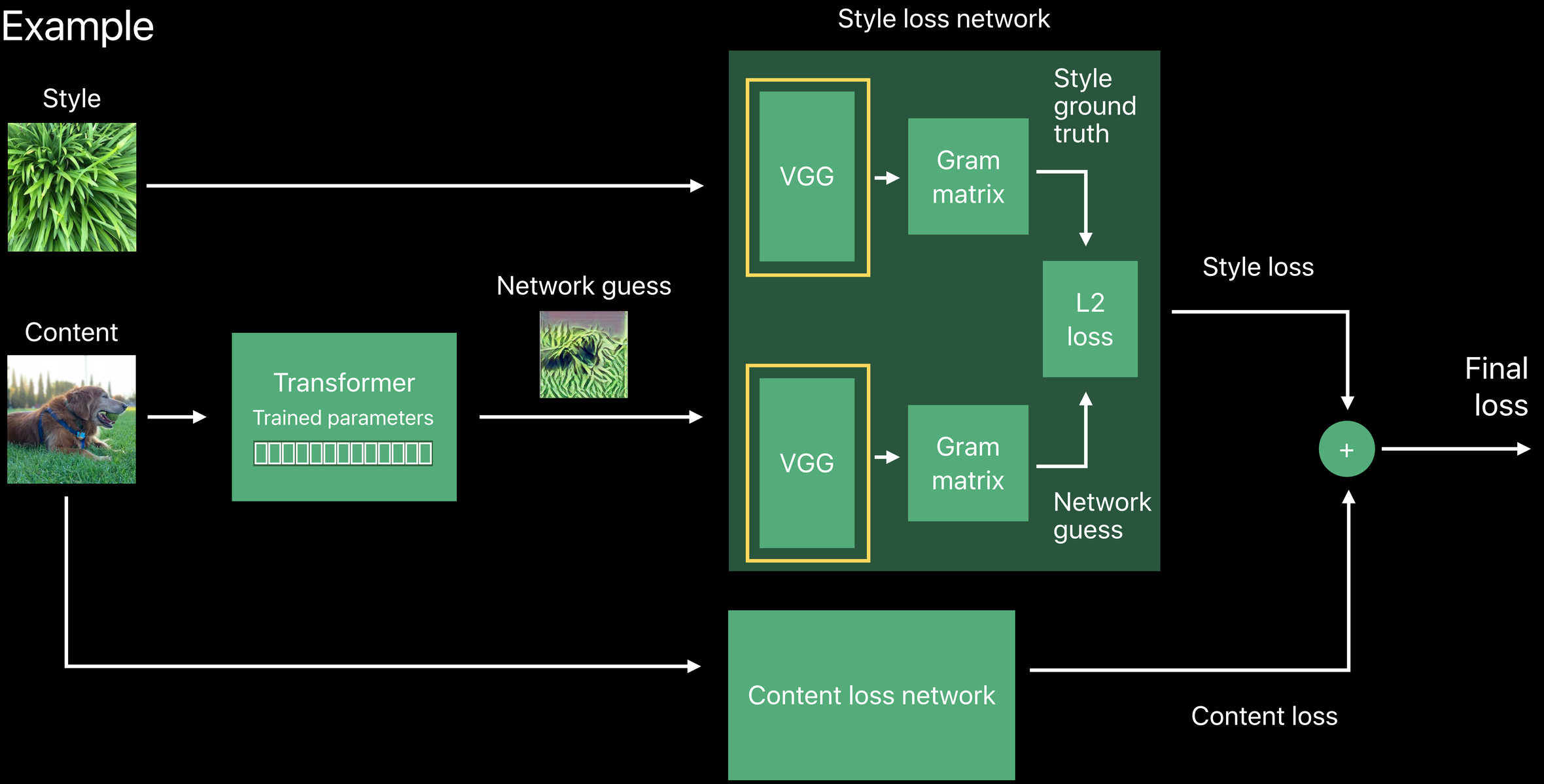
Style Transfer — Training

Example



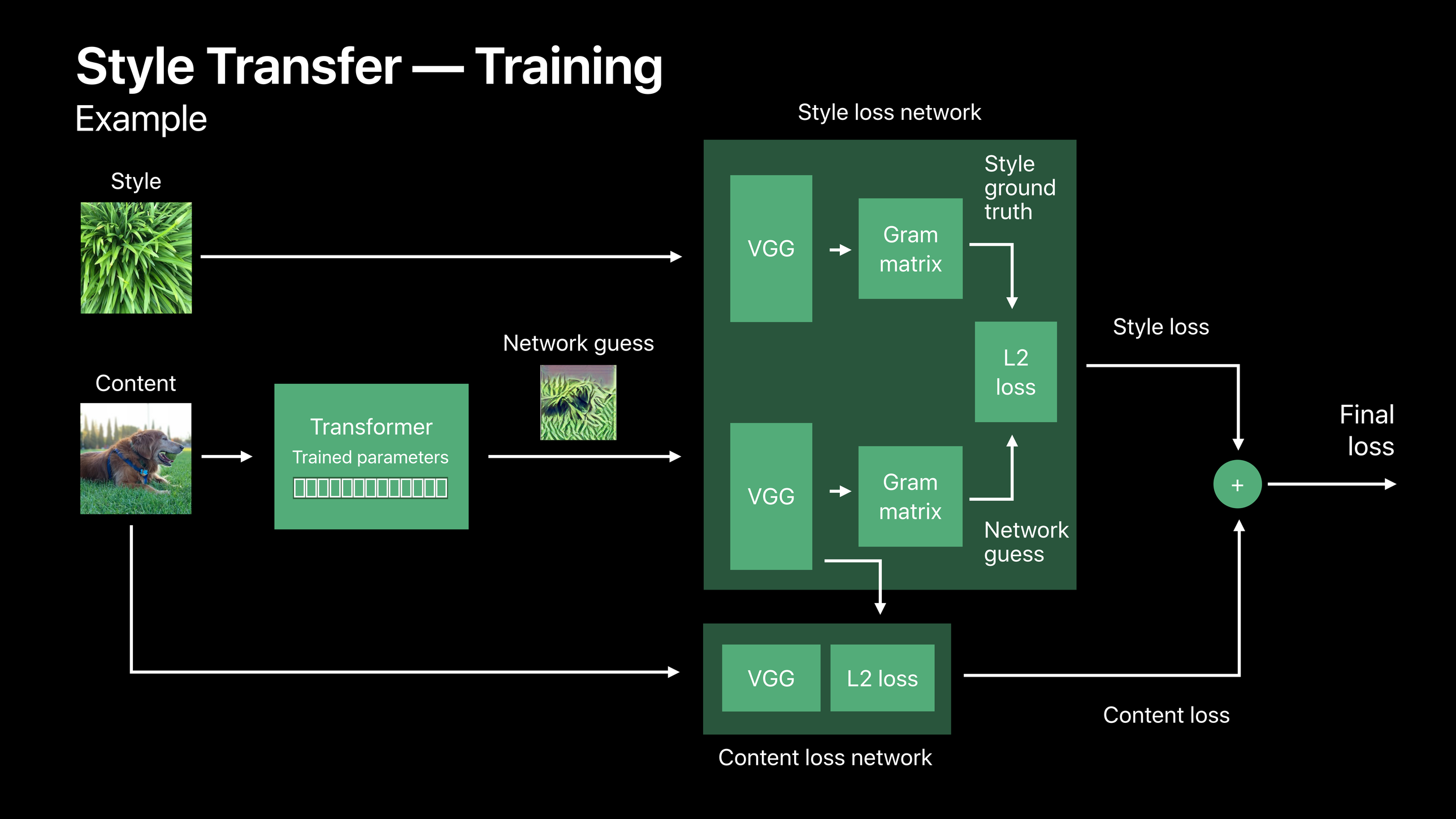
Style Transfer — Training

Example



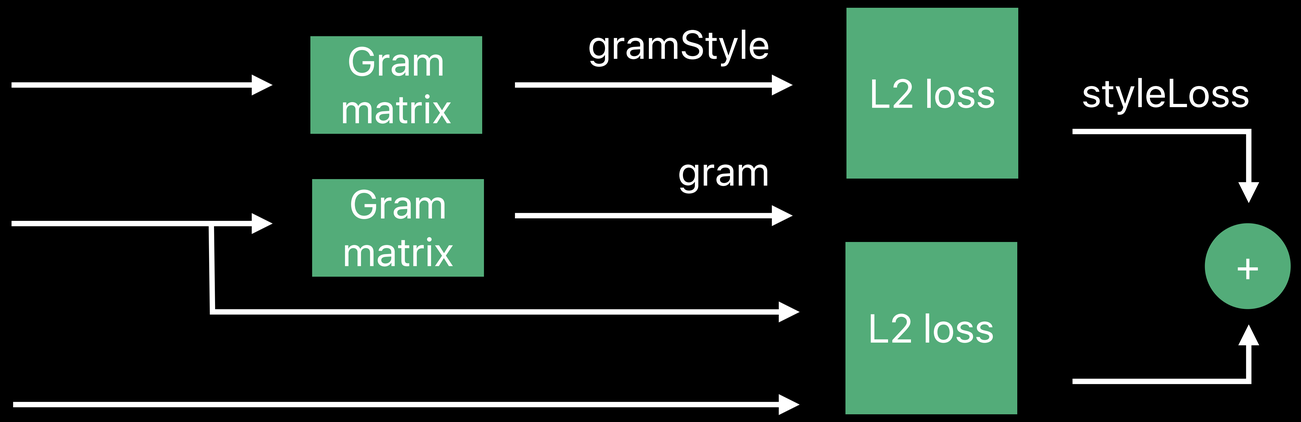
Style Transfer — Training

Example



Style Transfer — Training

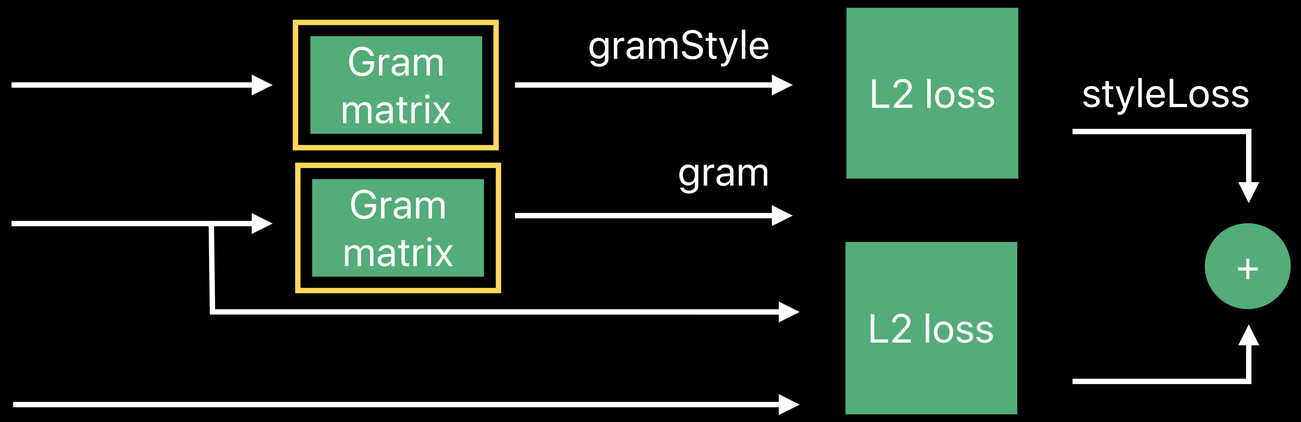
Example




```
let gramStyle = MPSNNGramMatrixCalculationNode(source: featuresStyle.resultImage)

let gram = MPSNNGramMatrixCalculationNode(source: features.resultImage)

let styleLoss = MPSNNForwardLossNode(source: gram.resultImage,
                                     labels: gramStyle.resultImage,
                                     lossDescriptor: lossDescriptor)
```

Style Transfer — Training

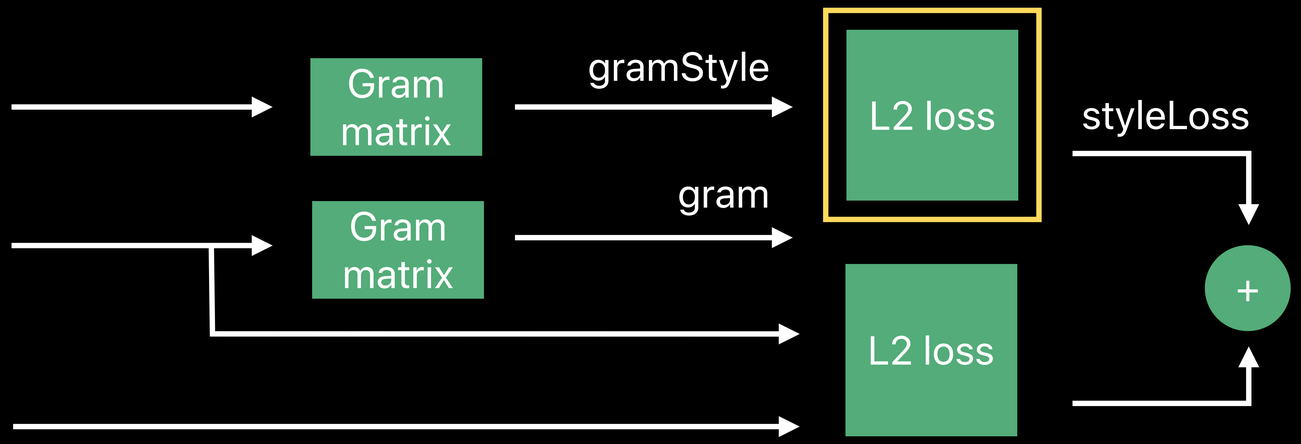
Example




```
let gramStyle = MPSNNGramMatrixCalculationNode(source: featuresStyle.resultImage)

let gram = MPSNNGramMatrixCalculationNode(source: features.resultImage)

let styleLoss = MPSNNForwardLossNode(source: gram.resultImage,
                                     labels: gramStyle.resultImage,
                                     lossDescriptor: lossDescriptor)
```

Style Transfer — Training

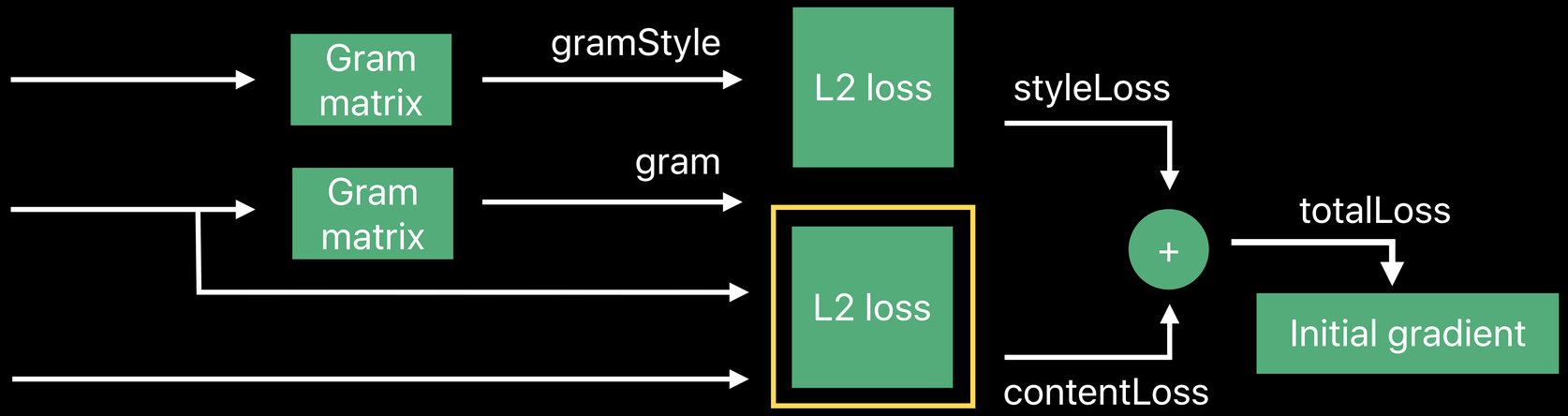
Example




```
let gramStyle = MPSNNGramMatrixCalculationNode(source: featuresStyle.resultImage)

let gram = MPSNNGramMatrixCalculationNode(source: features.resultImage)

let styleLoss = MPSNNForwardLossNode(source: gram.resultImage,
                                     labels: gramStyle.resultImage,
                                     lossDescriptor: lossDescriptor)
```

Style Transfer — Training

Example




```
let contentLoss = MPSNNForwardLossNode(source:features.resultImage,
                                       labels:featuresOriginal.resultImage,
                                       lossDescriptor:lossDescriptor)
let totalLoss = MPSNNAdditionNode(sources: [styleLoss.resultImage, contentLoss.resultImage])



let startGrad = MPSNNInitialGradientNode(source:totalLoss.resultImage)

let gradNodes = startGrad.trainingGraph(withSourceGradient: nil, nodeHandler: nil)
```

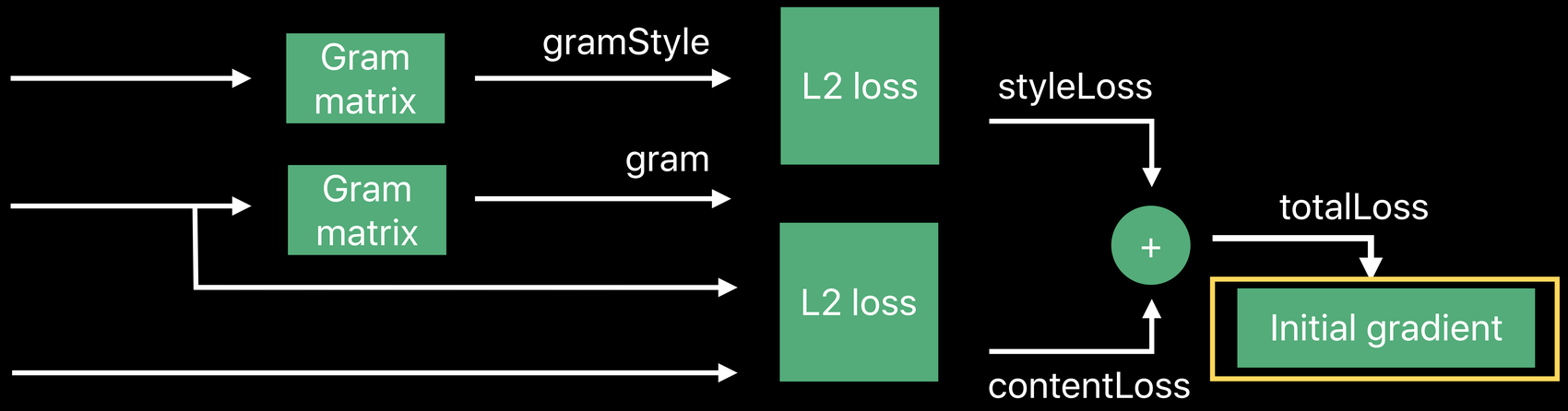
Style Transfer — Training

Example


```
let contentLoss = MPSNNForwardLossNode(source:features.resultImage,
                                       labels:featuresOriginal.resultImage,
                                       lossDescriptor:lossDescriptor)
let totalLoss = MPSNNAdditionNode(sources: [styleLoss.resultImage, contentLoss.resultImage])


let startGrad = MPSNNInitialGradientNode(source:totalLoss.resultImage)

let gradNodes = startGrad.trainingGraph(withSourceGradient: nil, nodeHandler: nil)
```

Style Transfer — Training

Example




```
let contentLoss = MPSNNForwardLossNode(source:features.resultImage,
                                       labels:featuresOriginal.resultImage,
                                       lossDescriptor:lossDescriptor)
let totalLoss = MPSNNAdditionNode(sources: [styleLoss.resultImage, contentLoss.resultImage])
```

```
let startGrad = MPSNNInitialGradientNode(source:totalLoss.resultImage)
```

```
let gradNodes = startGrad.trainingGraph(withSourceGradient: nil, nodeHandler: nil)
```

Style Transfer in Action

Style

Content

Iteration: 0

Stylized Image

Style Transfer in Action

Style

Content

Iteration: 0

Stylized Image

New Features

Implicit graph

Separable loss

Random number generation

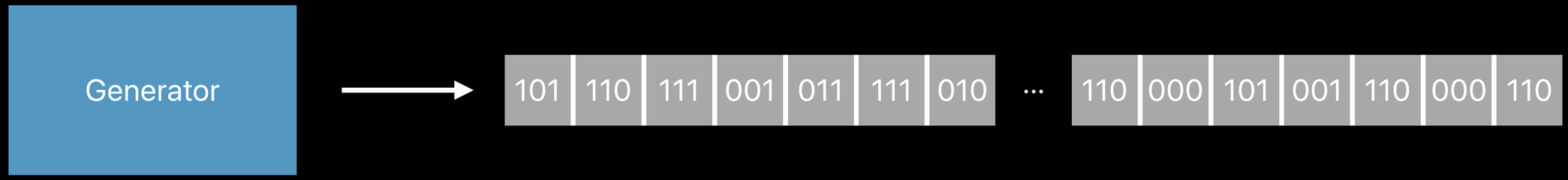
Predication

Commit and continue

Random Number Generation

Mersenne Twister (MTGP32)*

Philox**

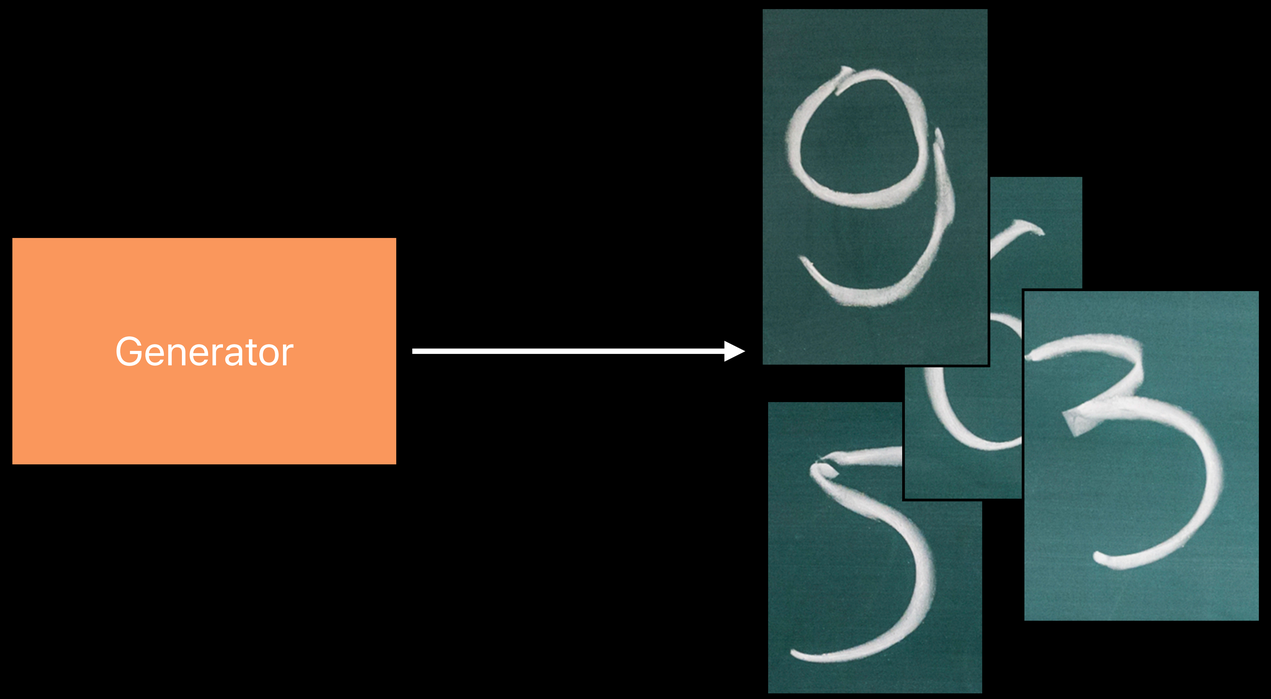


*Variants of Mersenne Twister Suitable for Graphics Processors, Mutsuo Saito, Makoto Matsomoto. https://arxiv.org/abs/1005.4973

**Parallel Random Numbers: As Easy as 1, 2, 3, John K. Salmon, Mark A. Moraes, Ron O. Dror, David E. Shaw.

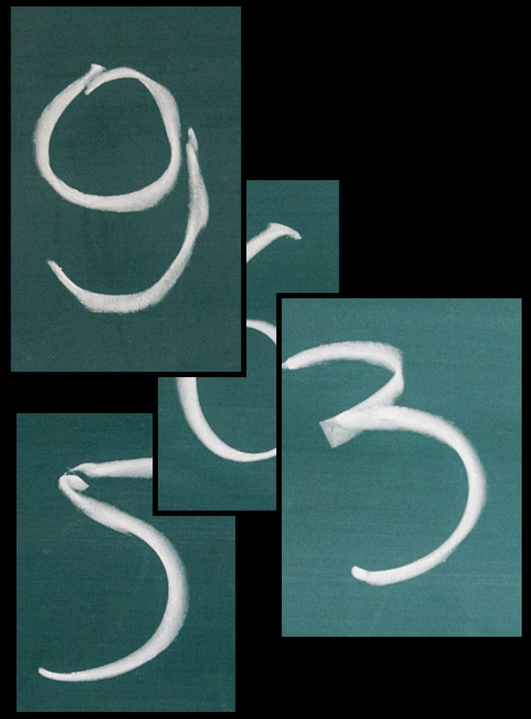
Generative Adversarial Network (GAN)

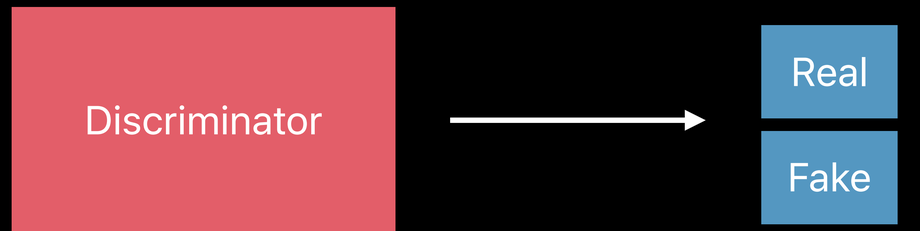
Example



Generative Adversarial Network (GAN)

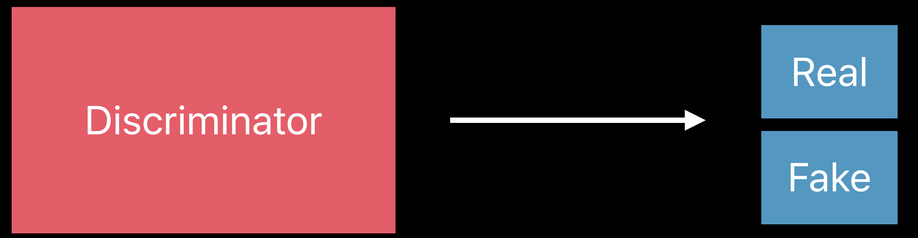
Example





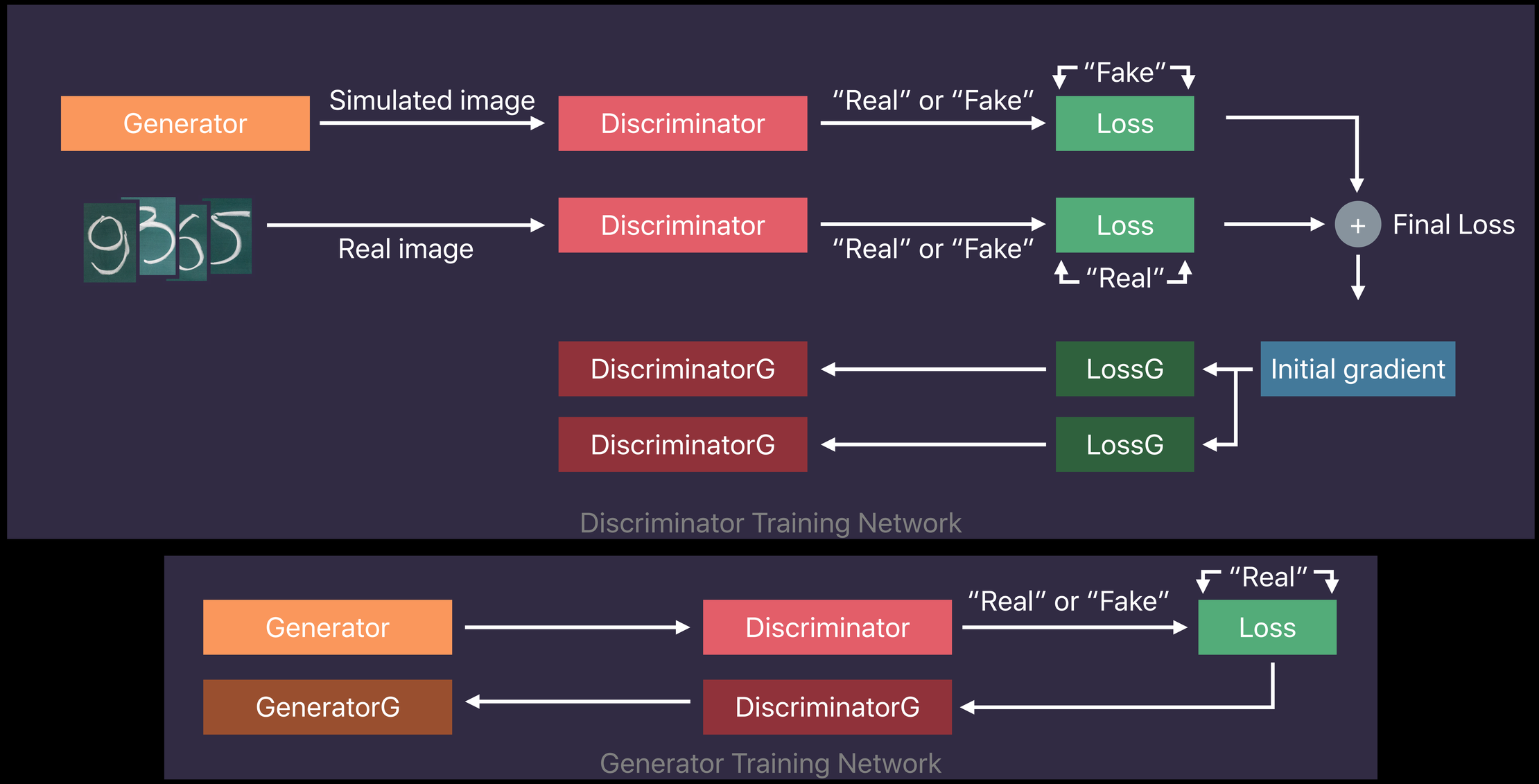
Generative Adversarial Network (GAN)

Example



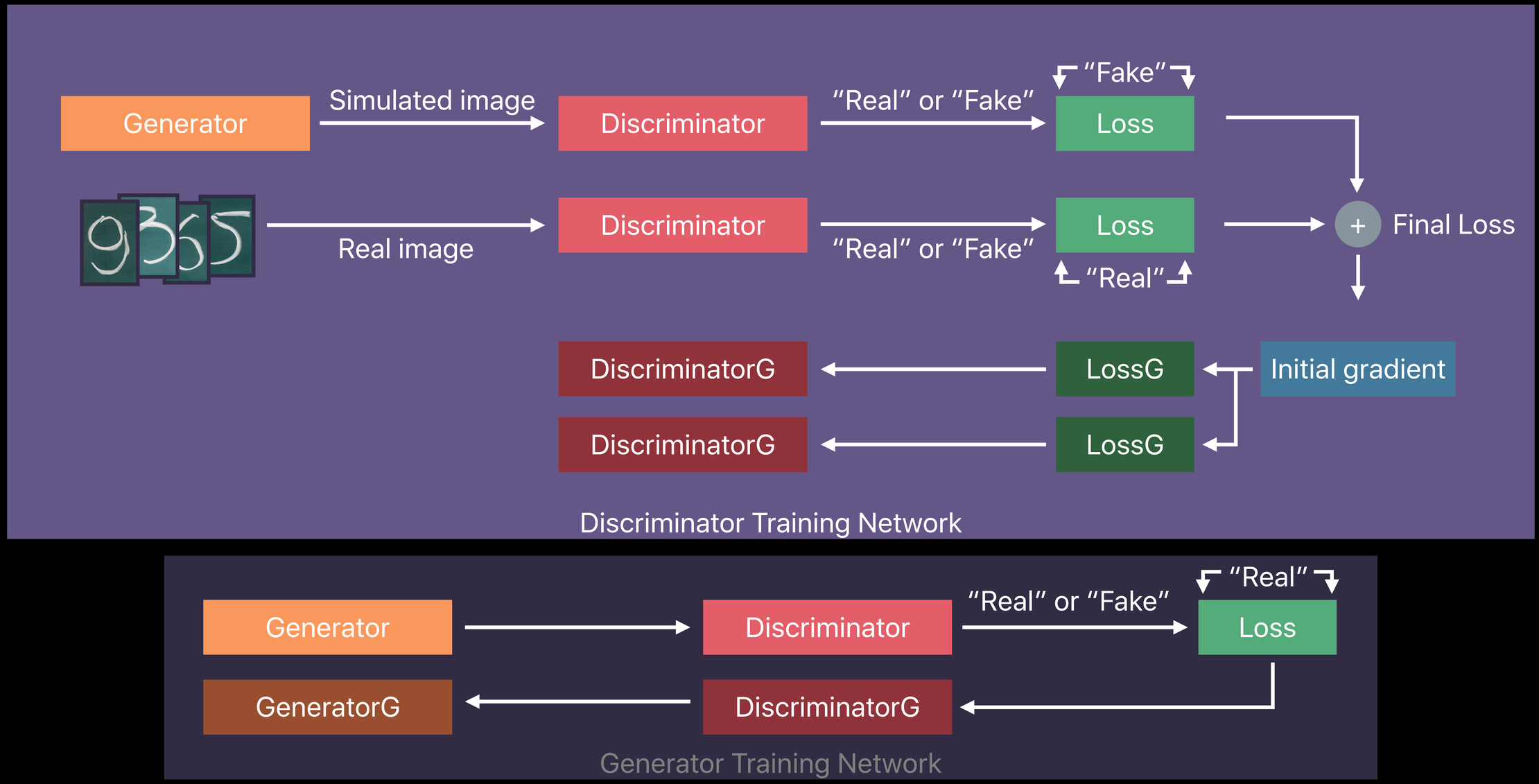
Generative Adversarial Network (GAN)

Example



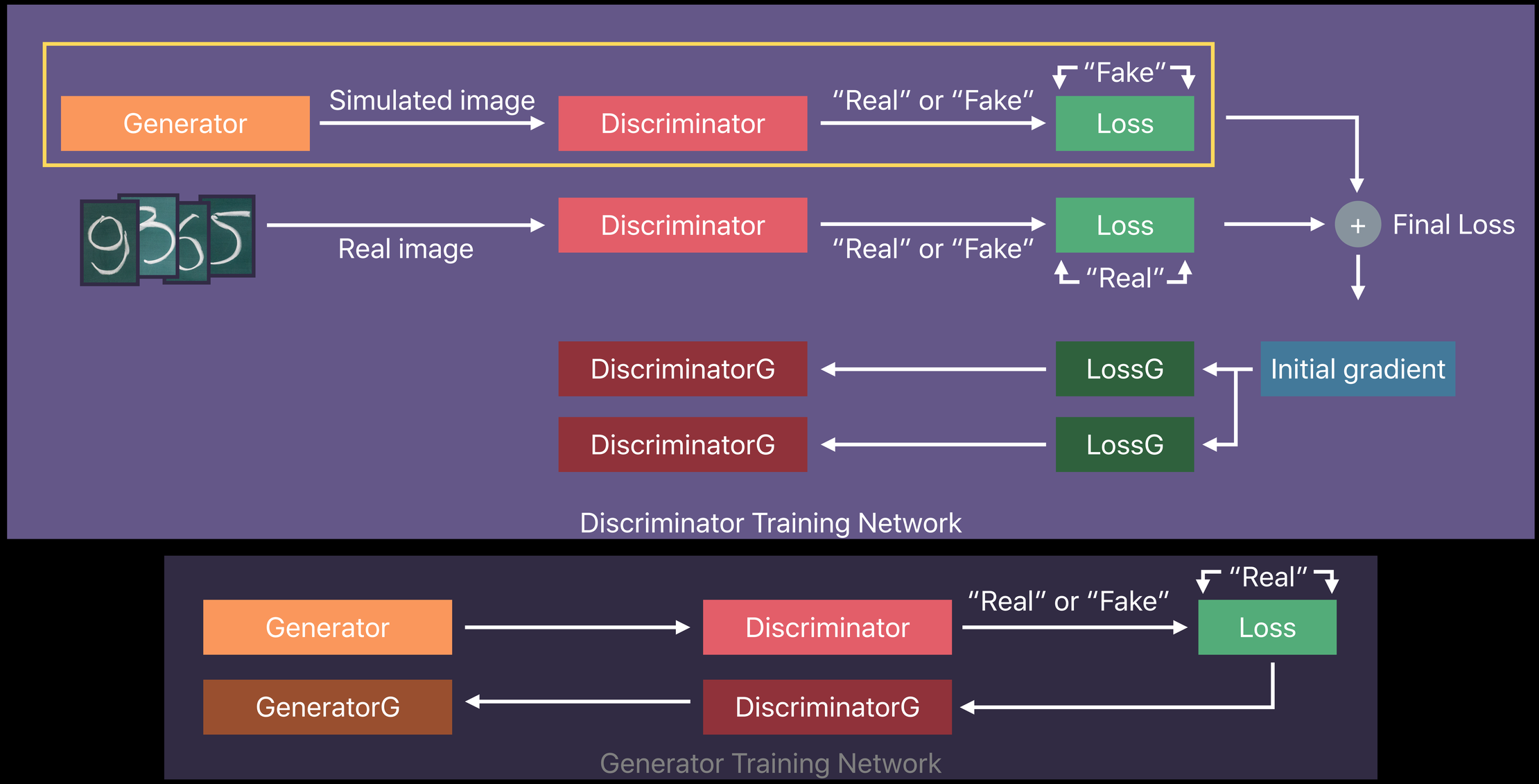
Generative Adversarial Network (GAN)

Example



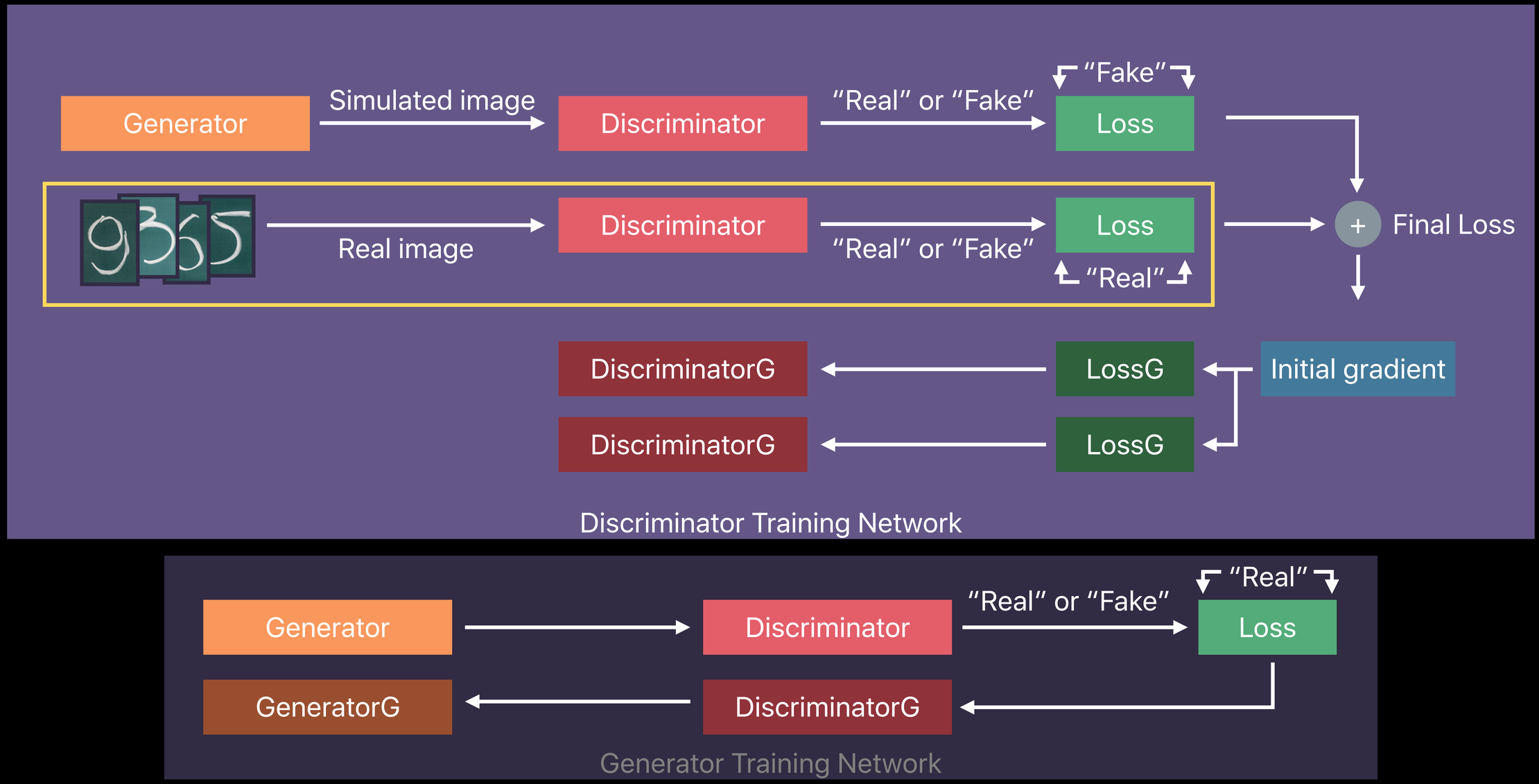
Generative Adversarial Network (GAN)

Example



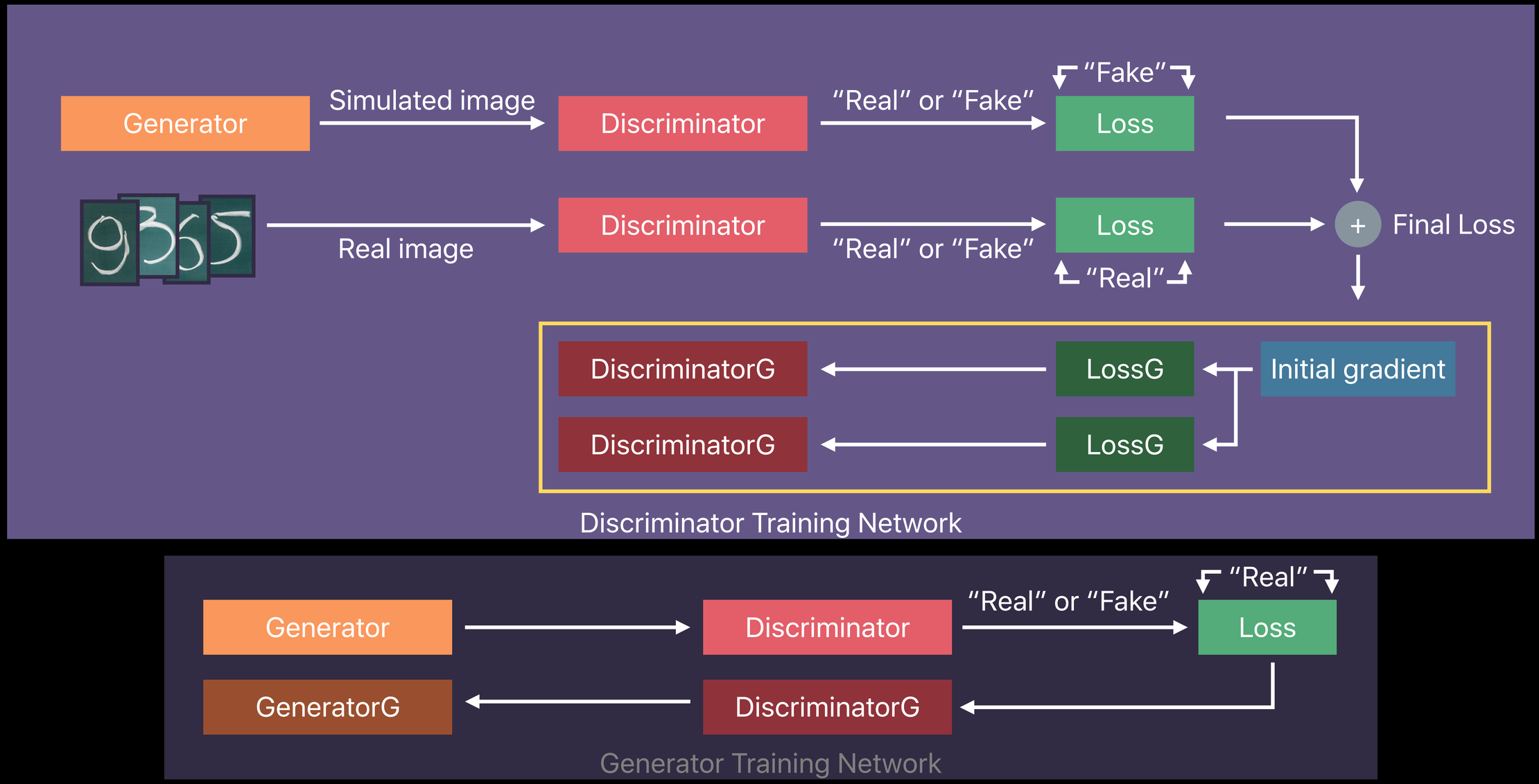
Generative Adversarial Network (GAN)

Example



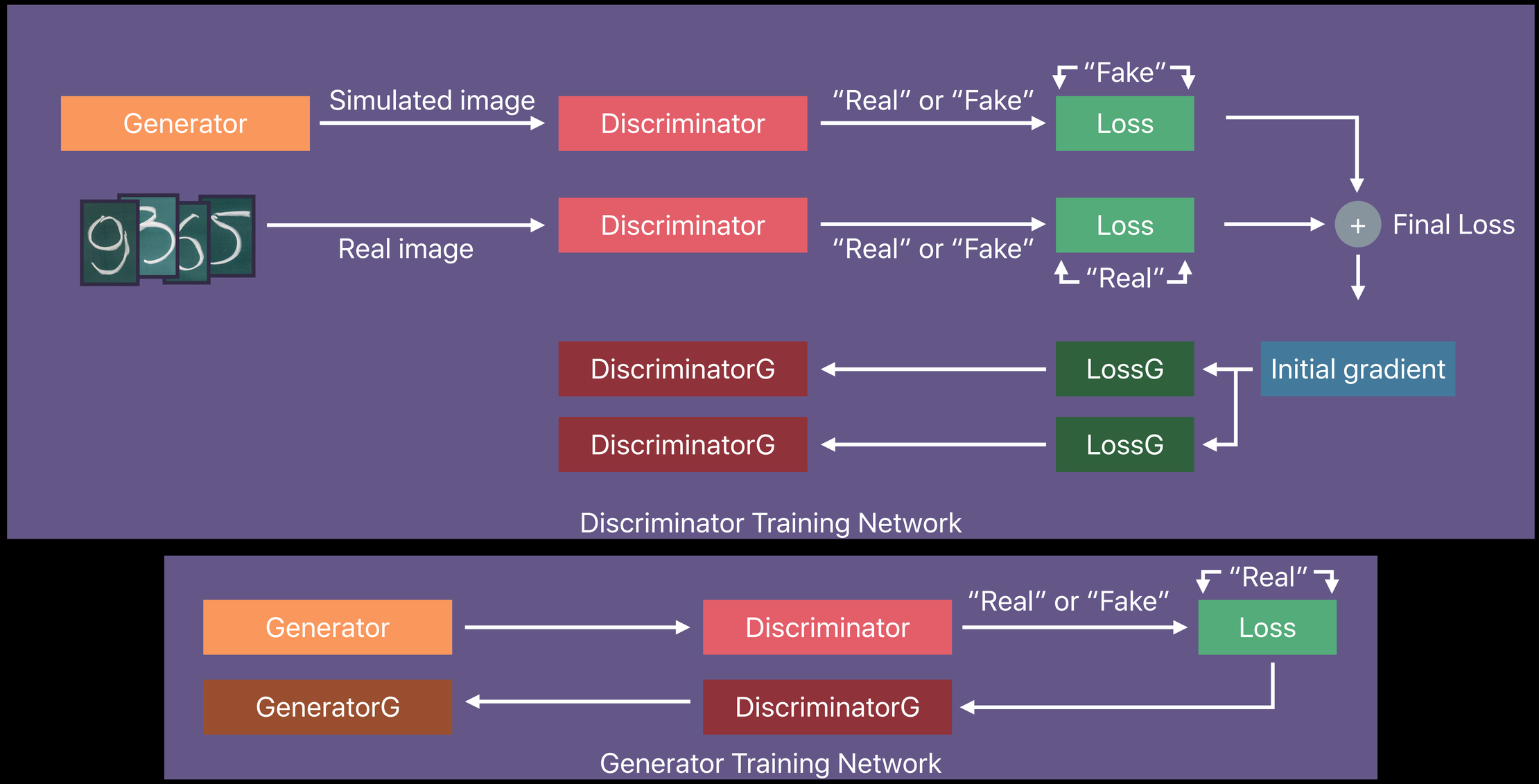
Generative Adversarial Network (GAN)

Example



Generative Adversarial Network (GAN)

Example



Random Number Generation

MPSMatrixRandom

Operates on MPSMatrix and MPSVector objects

MPSMatrixRandomDistributionDescriptor

```
let desc = MPSMatrixRandomDistributionDescriptor.uniformDistributionDescriptor(withMinimum: 0.0,
                                                                               maximum: 1.0)

let generator = MPSMatrixRandomMTGP32(device: device,
                                      destinationDataType: .float32,
                                      seed: 1234,
                                      distributionDescriptor: desc)
let result = MPSMatrix(device: device, descriptor: matDesc)
generator.encode(commandBuffer: cmdBuf, destinationMatrix: result)
```

Random Number Generation

MPSMatrixRandom

Operates on MPSMatrix and MPSVector objects

MPSMatrixRandomDistributionDescriptor

```
let desc = MPSMatrixRandomDistributionDescriptor.uniformDistributionDescriptor(withMinimum: 0.0,
                                                                               maximum: 1.0)

let generator = MPSMatrixRandomMTGP32(device: device,
                                      destinationDataType: .float32,
                                      seed: 1234,
                                      distributionDescriptor: desc)
let result = MPSMatrix(device: device, descriptor: matDesc)
generator.encode(commandBuffer: cmdBuf, destinationMatrix: result)
```

Random Number Generation

MPSMatrixRandom

Operates on MPSMatrix and MPSVector objects

MPSMatrixRandomDistributionDescriptor

```
let desc = MPSMatrixRandomDistributionDescriptor.uniformDistributionDescriptor(withMinimum: 0.0,
                                                                               maximum: 1.0)

let generator = MPSMatrixRandomMTGP32(device: device,
                                      destinationDataType: .float32,
                                      seed: 1234,
                                      distributionDescriptor: desc)
let result = MPSMatrix(device: device, descriptor: matDesc)
generator.encode(commandBuffer: cmdBuf, destinationMatrix: result)
```

Random Number Generation

MPSMatrixRandom

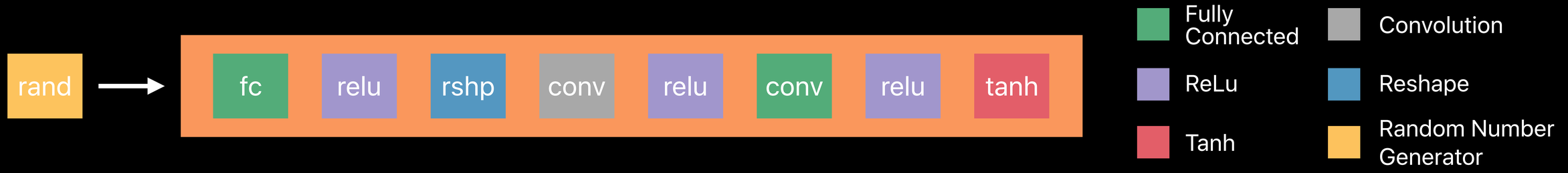
Operates on MPSMatrix and MPSVector objects

MPSMatrixRandomDistributionDescriptor

```
let desc = MPSMatrixRandomDistributionDescriptor.uniformDistributionDescriptor(withMinimum: 0.0,
                                                                               maximum: 1.0)

let generator = MPSMatrixRandomMTGP32(device: device,
                                      destinationDataType: .float32,
                                      seed: 1234,
                                      distributionDescriptor: desc)
let result = MPSMatrix(device: device, descriptor: matDesc)
generator.encode(commandBuffer: cmdBuf, destinationMatrix: result)
```

Generative Adversarial Network (GAN)

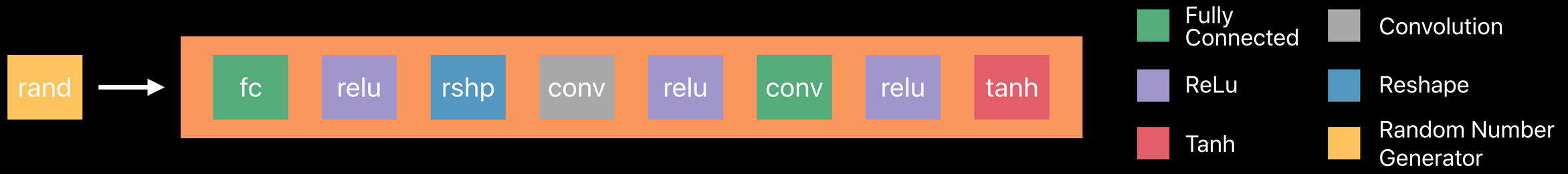
Example




```
let randMat = MPSMatrix(device: device, descriptor: matDesc)
let inputImage = MPSImage(device: device, imageDescriptor: imageDesc)


let copy = MPSMatrixCopyToImage(device: device, dataLayout: .HeightxWidthxFeatureChannels)


generator.encode(commandBuffer: cmdBuf, destinationMatrix: randMat)
copy.encode(commandBuffer: cmdBuf, sourceMatrix: randMat, destinationImage: inputImage)
graph.encode(to: cmdBuf, sourceImages: [inputImage])
```

Generative Adversarial Network (GAN)

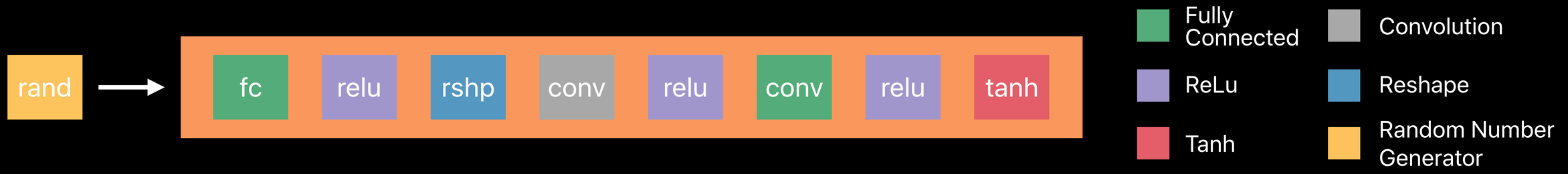
Example




```
let randMat = MPSMatrix(device: device, descriptor: matDesc)
let inputImage = MPSImage(device: device, imageDescriptor: imageDesc)


let copy = MPSMatrixCopyToImage(device: device, dataLayout: .HeightxWidthxFeatureChannels)


generator.encode(commandBuffer: cmdBuf, destinationMatrix: randMat)
copy.encode(commandBuffer: cmdBuf, sourceMatrix: randMat, destinationImage: inputImage)
graph.encode(to: cmdBuf, sourceImages: [inputImage])
```

Generative Adversarial Network (GAN)

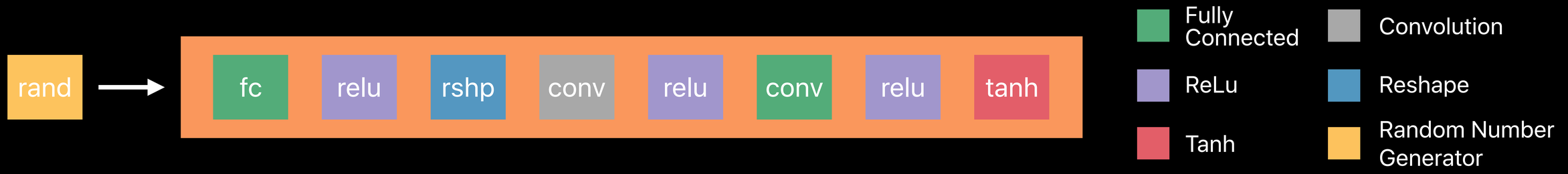
Example




```
let randMat = MPSMatrix(device: device, descriptor: matDesc)
let inputImage = MPSImage(device: device, imageDescriptor: imageDesc)

let copy = MPSMatrixCopyToImage(device: device, dataLayout: .HeightxWidthxFeatureChannels)

generator.encode(commandBuffer: cmdBuf, destinationMatrix: randMat)
copy.encode(commandBuffer: cmdBuf, sourceMatrix: randMat, destinationImage: inputImage)
graph.encode(to: cmdBuf, sourceImages: [inputImage])
```

Generative Adversarial Network (GAN)

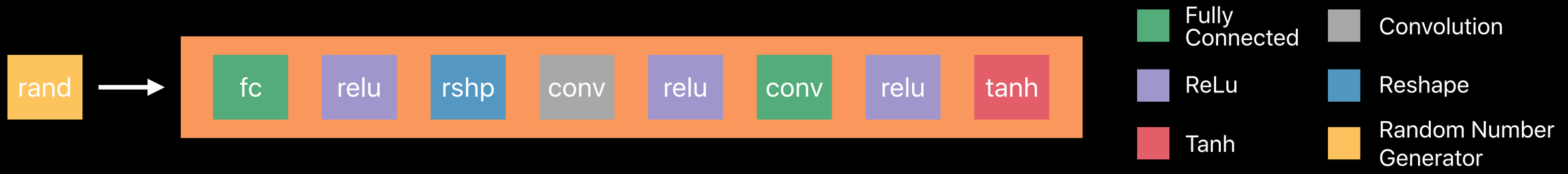
Example




```
let randMat = MPSMatrix(device: device, descriptor: matDesc)
let inputImage = MPSImage(device: device, imageDescriptor: imageDesc)

let copy = MPSMatrixCopyToImage(device: device, dataLayout: .HeightxWidthxFeatureChannels)

generator.encode(commandBuffer: cmdBuf, destinationMatrix: randMat)
copy.encode(commandBuffer: cmdBuf, sourceMatrix: randMat, destinationImage: inputImage)
graph.encode(to: cmdBuf, sourceImages: [inputImage])
```

Generative Adversarial Network (GAN)

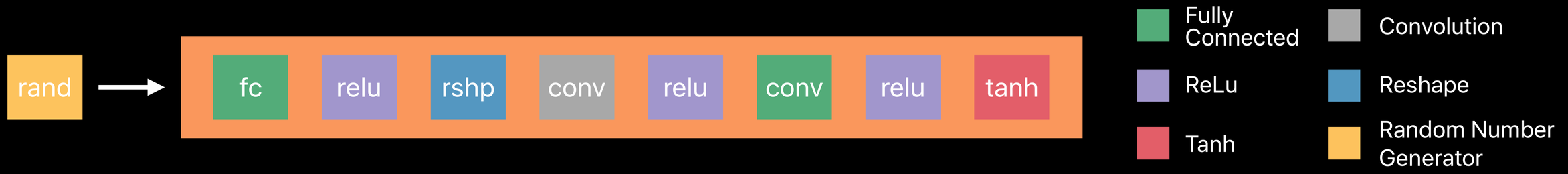
Example




```
let randMat = MPSMatrix(device: device, descriptor: matDesc)
let inputImage = MPSImage(device: device, imageDescriptor: imageDesc)


let copy = MPSMatrixCopyToImage(device: device, dataLayout: .HeightxWidthxFeatureChannels)


generator.encode(commandBuffer: cmdBuf, destinationMatrix: randMat)
copy.encode(commandBuffer: cmdBuf, sourceMatrix: randMat, destinationImage: inputImage)
graph.encode(to: cmdBuf, sourceImages: [inputImage])
```

Generative Adversarial Network (GAN)

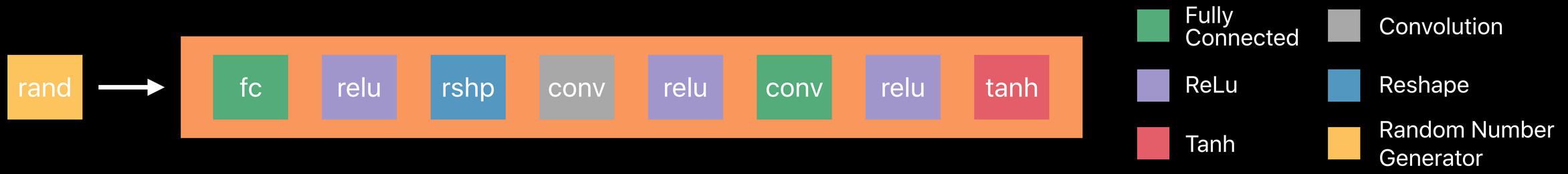
Example




```
let randMat = MPSMatrix(device: device, descriptor: matDesc)
let inputImage = MPSImage(device: device, imageDescriptor: imageDesc)


let copy = MPSMatrixCopyToImage(device: device, dataLayout: .HeightxWidthxFeatureChannels)


generator.encode(commandBuffer: cmdBuf, destinationMatrix: randMat)
copy.encode(commandBuffer: cmdBuf, sourceMatrix: randMat, destinationImage: inputImage)
graph.encode(to: cmdBuf, sourceImages: [inputImage])
```

Generative Adversarial Network (GAN)

Example




```
let randMat = MPSMatrix(device: device, descriptor: matDesc)
let inputImage = MPSImage(device: device, imageDescriptor: imageDesc)


let copy = MPSMatrixCopyToImage(device: device, dataLayout: .HeightxWidthxFeatureChannels)


generator.encode(commandBuffer: cmdBuf, destinationMatrix: randMat)
copy.encode(commandBuffer: cmdBuf, sourceMatrix: randMat, destinationImage: inputImage)
graph.encode(to: cmdBuf, sourceImages: [inputImage])
```

New Features

Implicit graph

Separable loss

Random number generation

Predication

Commit and continue

Predication

Conditional execution of MPS kernels

Predicate value used at execution time

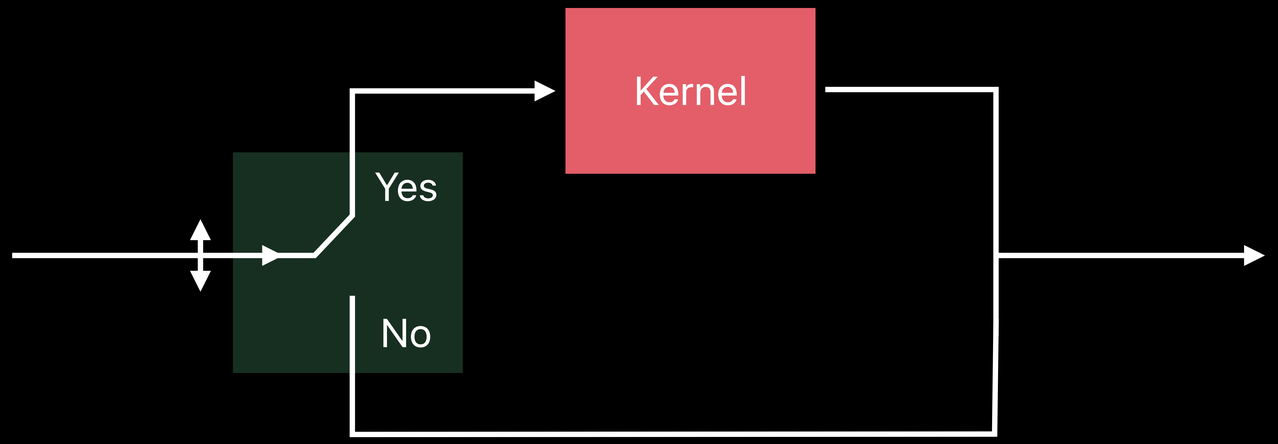
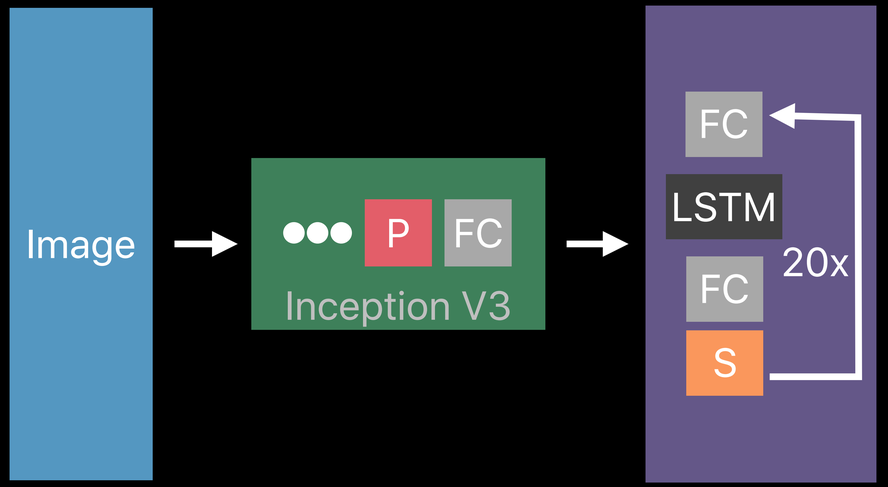


Image Captioning

Example

Image Captioning — Inference

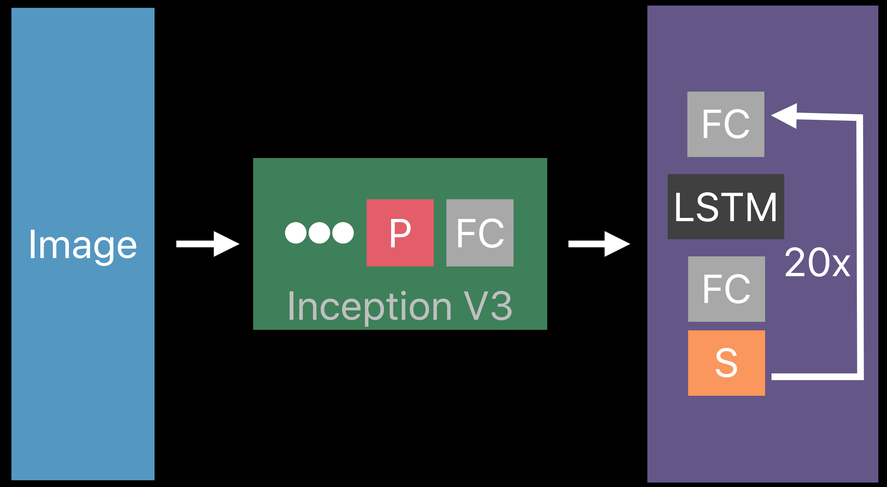
Example



P Max Pooling FC Fully Connected S SoftMax LSTM LSTM

Image Captioning — Inference

Example



P Max Pooling FC Fully Connected S SoftMax LSTM LSTM

Image Captioning — Inference

Example

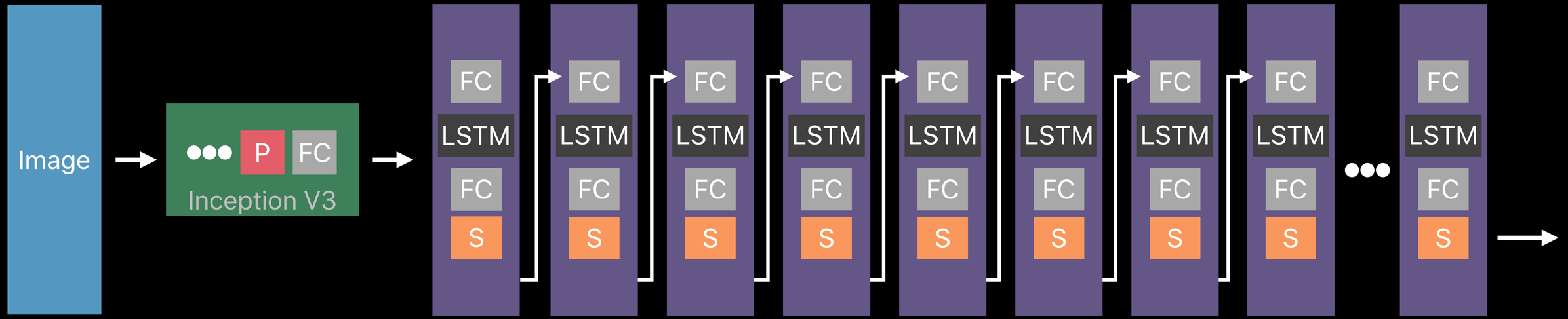


Image Captioning — Inference

Example

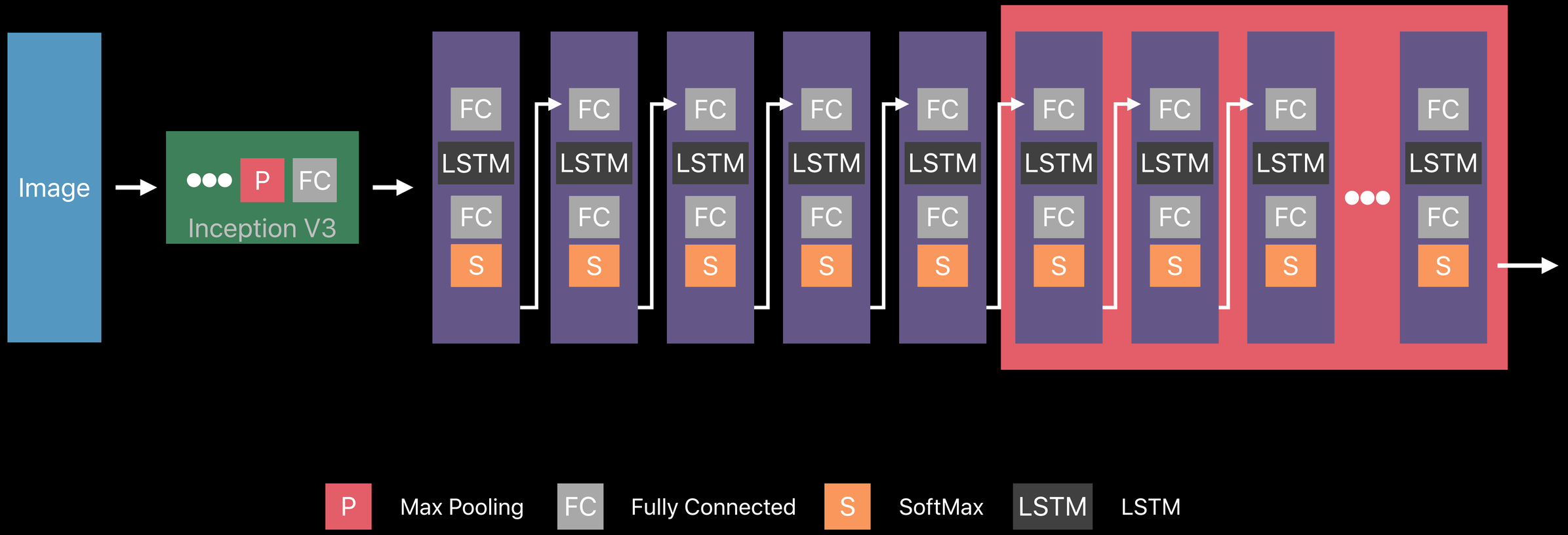
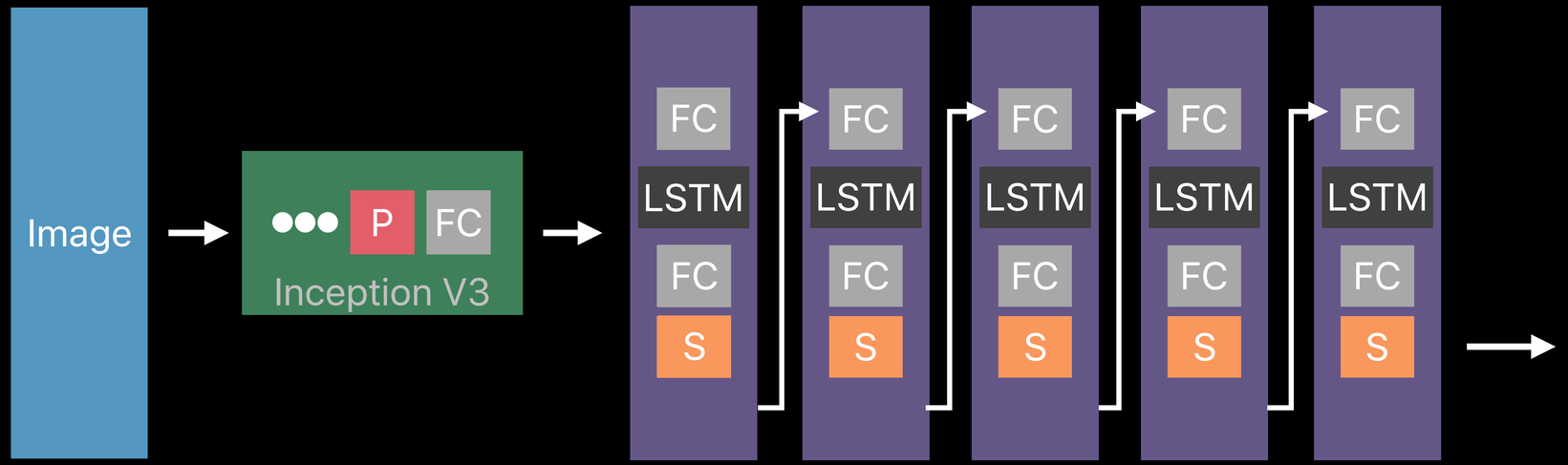


Image Captioning — Inference

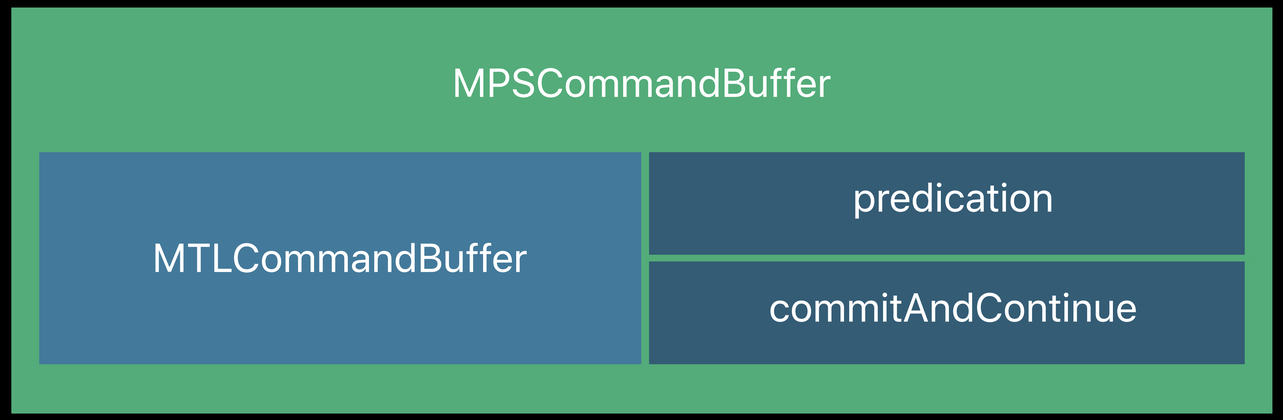
Example



MPSCommandBuffer

Conforms to MTLCommandBuffer protocol

Create from a MTLCommandQueue

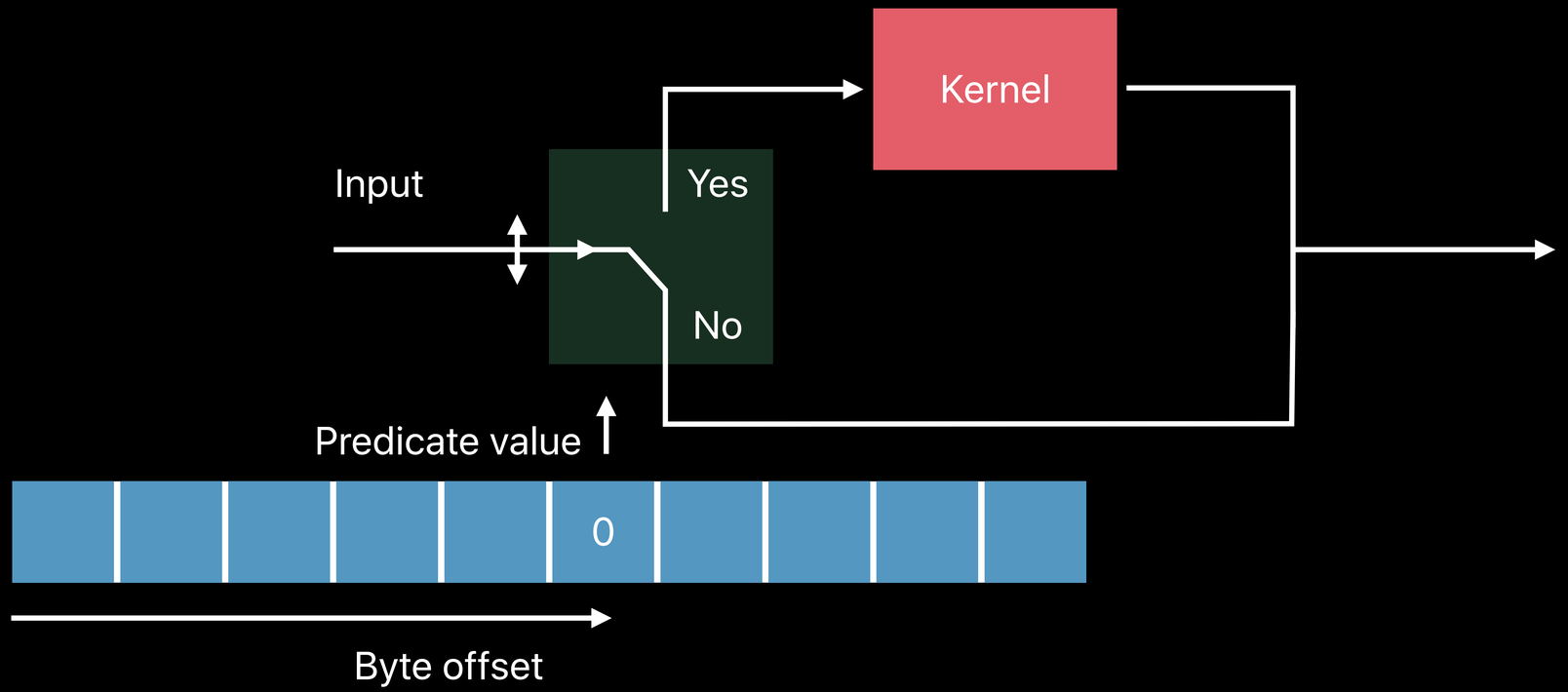


MPSPredicate

Predicate contained in MTLBuffer

Byte offset locates predicate value

Create from MTLBuffer or MTLDevice



```
// Create, set, and use an MPSPredicate object
let predicate = MPSPredicate(device: device)

mpsCommandBuffer.predicate = predicate

// Wrap the predicate buffer in an MPSMatrix
let predicateMatrix = MPSMatrix(buffer: predicate.predicateBuffer,
                                offset: predicate.predicateOffset,
                            descriptor: predicateMatDesc)

// Define a linear neuron to act as a decrementer
matNeuron.setNeuronType(.linear, parameterA: 1.0, parameterB: -1.0, parameterC: 0.0)
matNeuron.encode(commandBuffer: mpsCommandBuffer,
                   inputMatrix: predicateMatrix,
                    biasVector: nil,
                  resultMatrix: predicateMatrix)
let result = cnnKernel.encode(commandBuffer: mpsCommandBuffer,
                              sourceImage: inputImage)
```

```
// Create, set, and use an MPSPredicate object
let predicate = MPSPredicate(device: device)

mpsCommandBuffer.predicate = predicate

// Wrap the predicate buffer in an MPSMatrix
let predicateMatrix = MPSMatrix(buffer: predicate.predicateBuffer,
                                offset: predicate.predicateOffset,
                            descriptor: predicateMatDesc)

// Define a linear neuron to act as a decrementer
matNeuron.setNeuronType(.linear, parameterA: 1.0, parameterB: -1.0, parameterC: 0.0)
matNeuron.encode(commandBuffer: mpsCommandBuffer,
                   inputMatrix: predicateMatrix,
                    biasVector: nil,
                  resultMatrix: predicateMatrix)
let result = cnnKernel.encode(commandBuffer: mpsCommandBuffer,
                              sourceImage: inputImage)
```

```
// Create, set, and use an MPSPredicate object
let predicate = MPSPredicate(device: device)

mpsCommandBuffer.predicate = predicate

// Wrap the predicate buffer in an MPSMatrix
let predicateMatrix = MPSMatrix(buffer: predicate.predicateBuffer,
                                offset: predicate.predicateOffset,
                            descriptor: predicateMatDesc)

// Define a linear neuron to act as a decrementer
matNeuron.setNeuronType(.linear, parameterA: 1.0, parameterB: -1.0, parameterC: 0.0)
matNeuron.encode(commandBuffer: mpsCommandBuffer,
                   inputMatrix: predicateMatrix,
                    biasVector: nil,
                  resultMatrix: predicateMatrix)
let result = cnnKernel.encode(commandBuffer: mpsCommandBuffer,
                              sourceImage: inputImage)
```

```
// Create, set, and use an MPSPredicate object
let predicate = MPSPredicate(device: device)

mpsCommandBuffer.predicate = predicate

// Wrap the predicate buffer in an MPSMatrix
let predicateMatrix = MPSMatrix(buffer: predicate.predicateBuffer,
                                offset: predicate.predicateOffset,
                            descriptor: predicateMatDesc)

// Define a linear neuron to act as a decrementer
matNeuron.setNeuronType(.linear, parameterA: 1.0, parameterB: -1.0, parameterC: 0.0)
matNeuron.encode(commandBuffer: mpsCommandBuffer,
                   inputMatrix: predicateMatrix,
                    biasVector: nil,
                  resultMatrix: predicateMatrix)
let result = cnnKernel.encode(commandBuffer: mpsCommandBuffer,
                              sourceImage: inputImage)
```

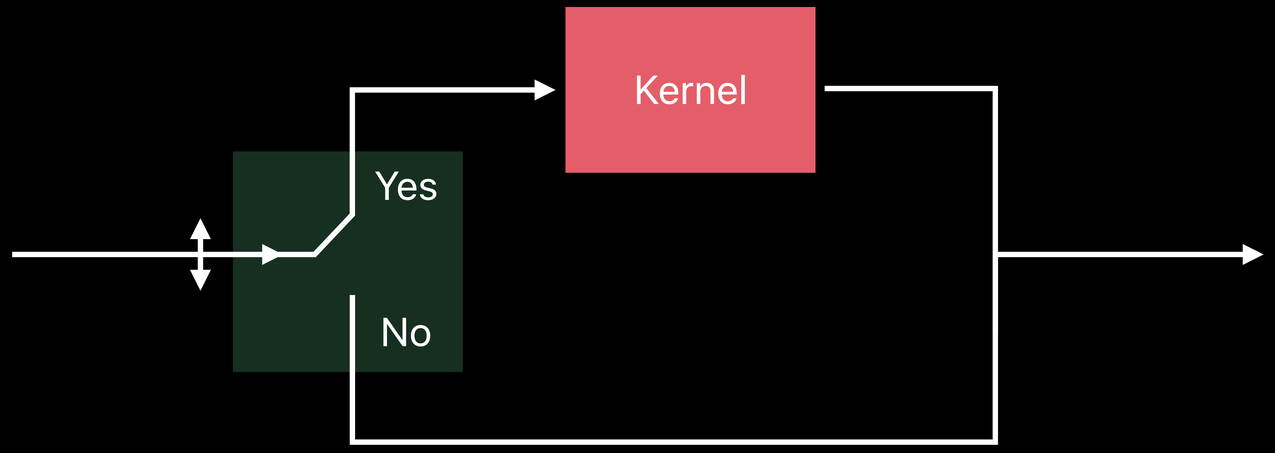
```
// Create, set, and use an MPSPredicate object
let predicate = MPSPredicate(device: device)

mpsCommandBuffer.predicate = predicate

// Wrap the predicate buffer in an MPSMatrix
let predicateMatrix = MPSMatrix(buffer: predicate.predicateBuffer,
                                offset: predicate.predicateOffset,
                            descriptor: predicateMatDesc)

// Define a linear neuron to act as a decrementer
matNeuron.setNeuronType(.linear, parameterA: 1.0, parameterB: -1.0, parameterC: 0.0)
matNeuron.encode(commandBuffer: mpsCommandBuffer,
                   inputMatrix: predicateMatrix,
                    biasVector: nil,
                  resultMatrix: predicateMatrix)
let result = cnnKernel.encode(commandBuffer: mpsCommandBuffer,
                              sourceImage: inputImage)
```

```
// Create, set, and use an MPSPredicate object
let predicate = MPSPredicate(device: device)

mpsCommandBuffer.predicate = predicate

// Wrap the predicate buffer in an MPSMatrix
let predicateMatrix = MPSMatrix(buffer: predicate.predicateBuffer,
                                offset: predicate.predicateOffset,
                            descriptor: predicateMatDesc)

// Define a linear neuron to act as a decrementer
matNeuron.setNeuronType(.linear, parameterA: 1.0, parameterB: -1.0, parameterC: 0.0)
matNeuron.encode(commandBuffer: mpsCommandBuffer,
                   inputMatrix: predicateMatrix,
                    biasVector: nil,
                  resultMatrix: predicateMatrix)
let result = cnnKernel.encode(commandBuffer: mpsCommandBuffer,
                              sourceImage: inputImage)
```

Predication

Bypass kernels if possible

Share MTLBuffers among MPSPredicates



Predication

Bypass kernels if possible

Share MTLBuffers among MPSPredicates

New Features

Implicit graph

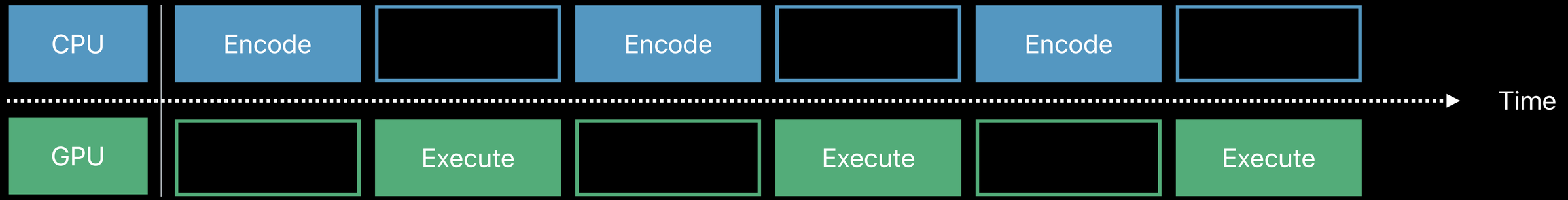
Separable loss

Random number generation

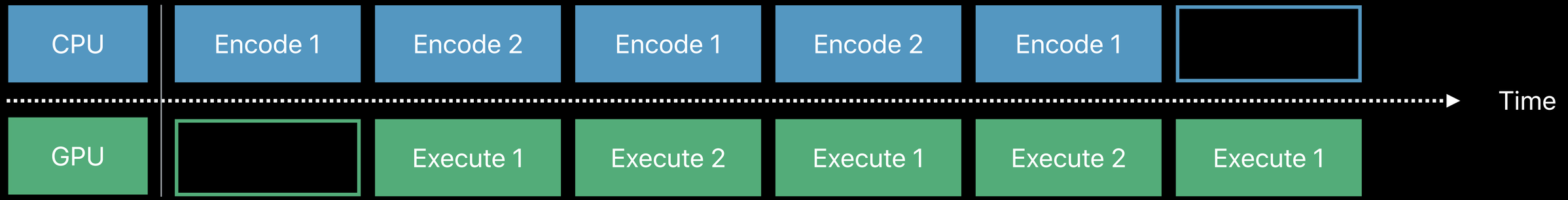
Predication

Commit and continue

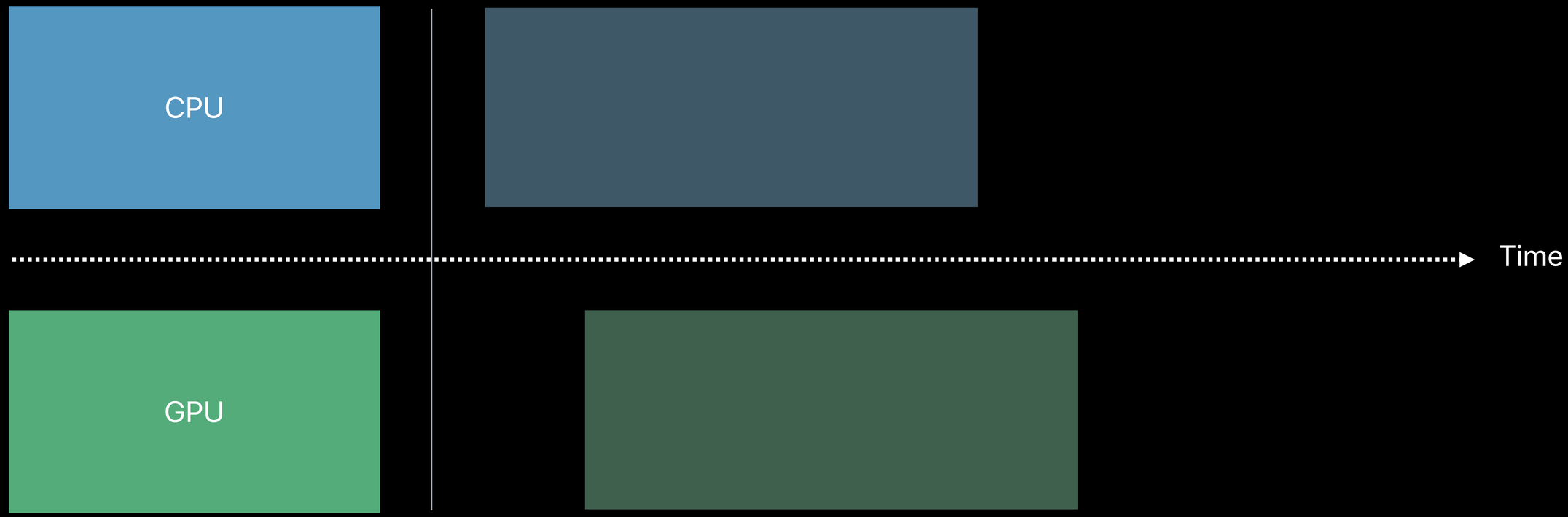
Single Buffering



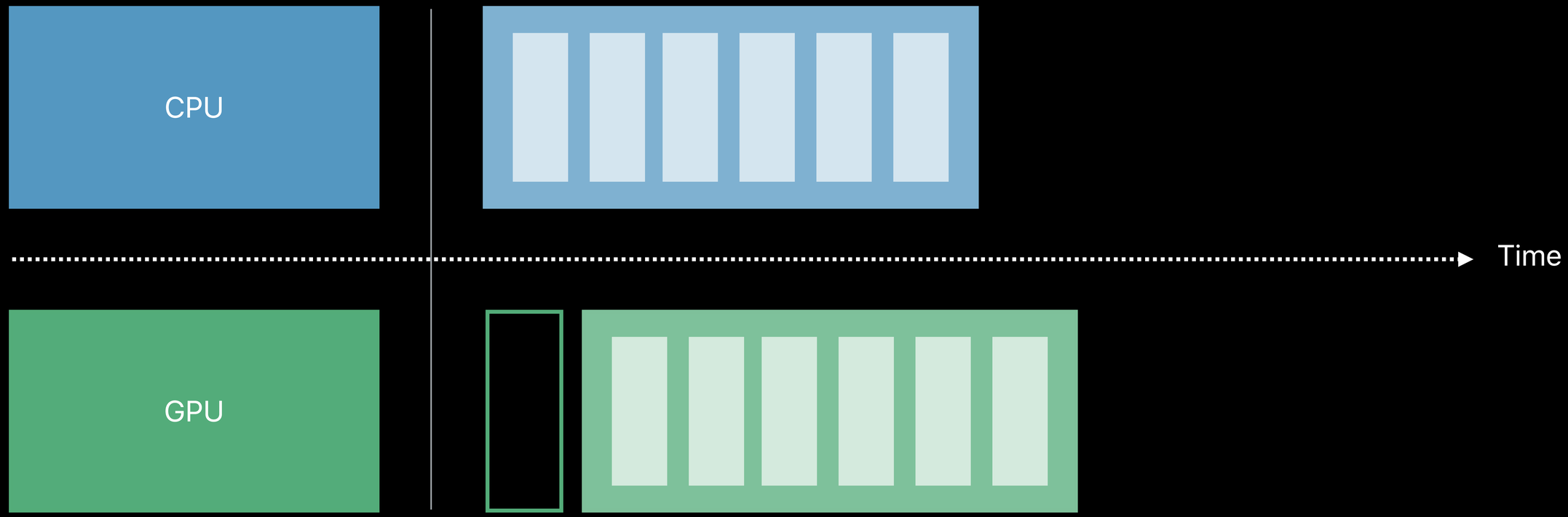
Double Buffering



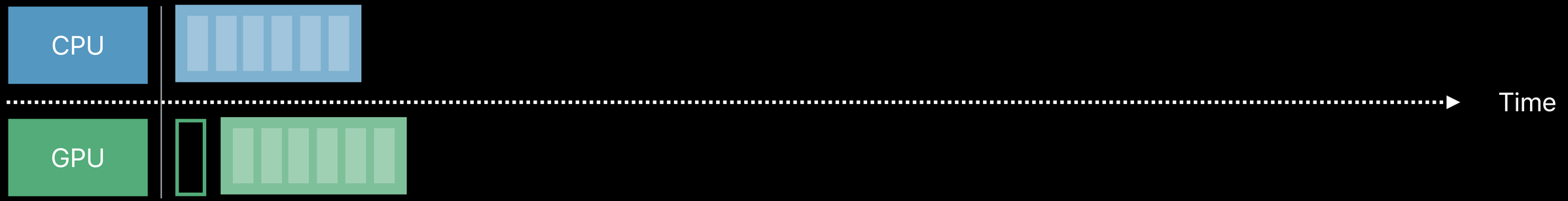
commitAndContinue



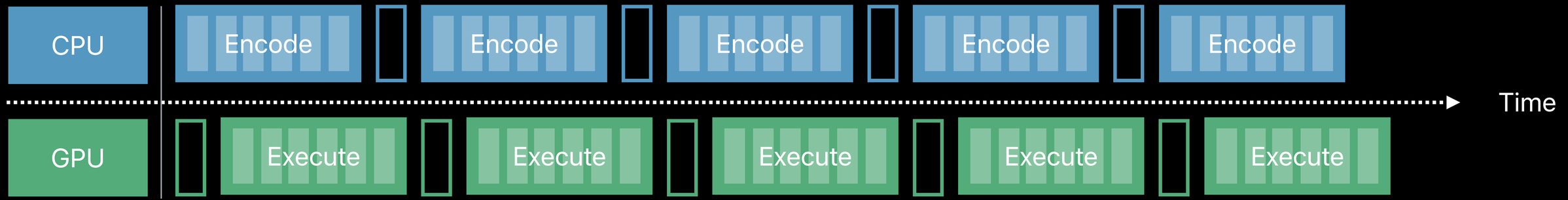
commitAndContinue



commitAndContinue



commitAndContinue



```
// Commit work using on MTLCommandBuffer
let cmdBuf = queue.makeCommandBuffer()!

cnnKernel0.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cnnKernel1.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cnnKernel2.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cnnKernel3.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cmdBuf.commit()
```

```
// Commit work using on MTLCommandBuffer
let cmdBuf = queue.makeCommandBuffer()!

cnnKernel0.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cnnKernel1.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cnnKernel2.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cnnKernel3.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cmdBuf.commit()
```

```
// Commit work using on MTLCommandBuffer
let cmdBuf = queue.makeCommandBuffer()!

cnnKernel0.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cnnKernel1.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cnnKernel2.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cnnKernel3.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cmdBuf.commit()
```

```
// Commit work using MPSCommandBuffer and commitAndContinue
let cmdBuf = MPSCommandBuffer(from: queue)

cnnKernel0.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cnnKernel1.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cmdBuf.commitAndContinue()

cnnKernel2.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cnnKernel3.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cmdBuf.commit()
```

```
// Commit work using MPSCommandBuffer and commitAndContinue
let cmdBuf = MPSCommandBuffer(from: queue)

cnnKernel0.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cnnKernel1.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cmdBuf.commitAndContinue()

cnnKernel2.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cnnKernel3.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cmdBuf.commit()
```

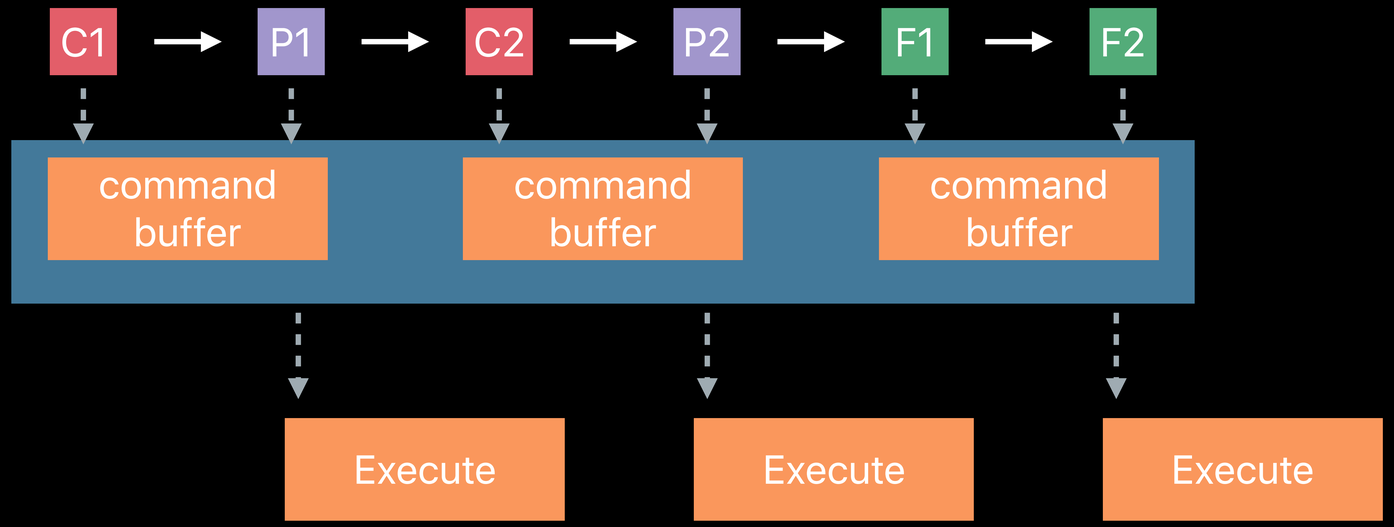
```
// Commit work using MPSCommandBuffer and commitAndContinue
let cmdBuf = MPSCommandBuffer(from: queue)

cnnKernel0.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cnnKernel1.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cmdBuf.commitAndContinue()

cnnKernel2.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cnnKernel3.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cmdBuf.commit()
```

```
// Commit work using MPSCommandBuffer and commitAndContinue
let cmdBuf = MPSCommandBuffer(from: queue)

cnnKernel0.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cnnKernel1.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cmdBuf.commitAndContinue()

cnnKernel2.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cnnKernel3.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cmdBuf.commit()
```

```
// Commit work using MPSCommandBuffer and commitAndContinue
let cmdBuf = MPSCommandBuffer(from: queue)

cnnKernel0.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cnnKernel1.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cmdBuf.commitAndContinue()

cnnKernel2.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cnnKernel3.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cmdBuf.commit()
```

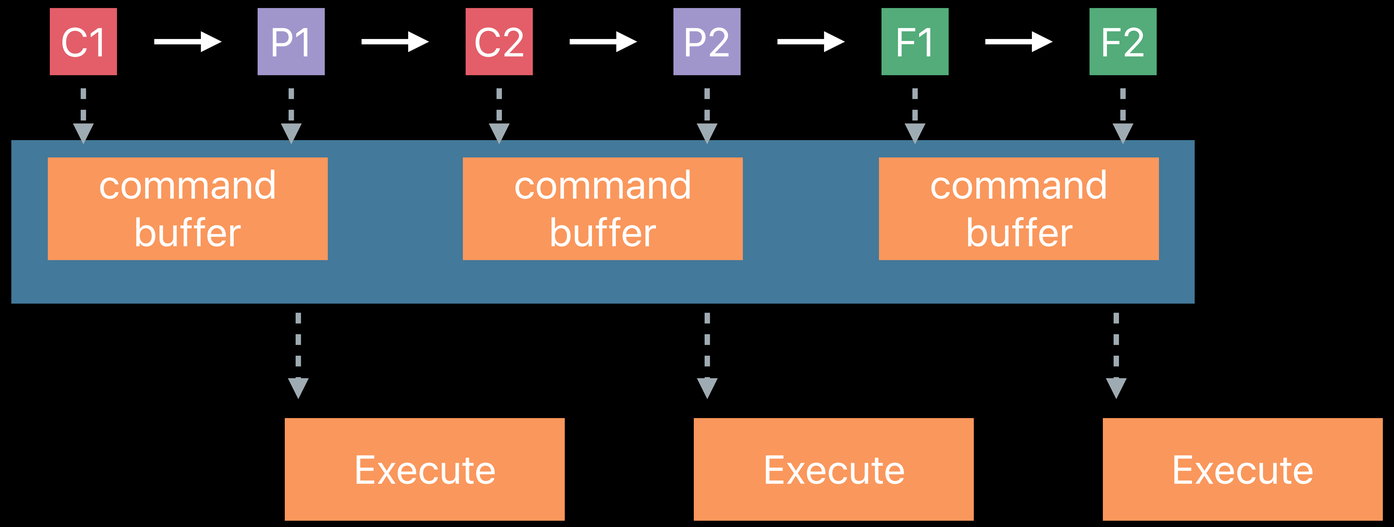
```
// Commit work using MPSCommandBuffer and commitAndContinue
let cmdBuf = MPSCommandBuffer(from: queue)

cnnKernel0.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cnnKernel1.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cmdBuf.commitAndContinue()

cnnKernel2.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cnnKernel3.encodeBatch(commandBuffer: cmdBuf, sourceImages: sources, destinationImages: results)

cmdBuf.commit()
```

commitAndContinue



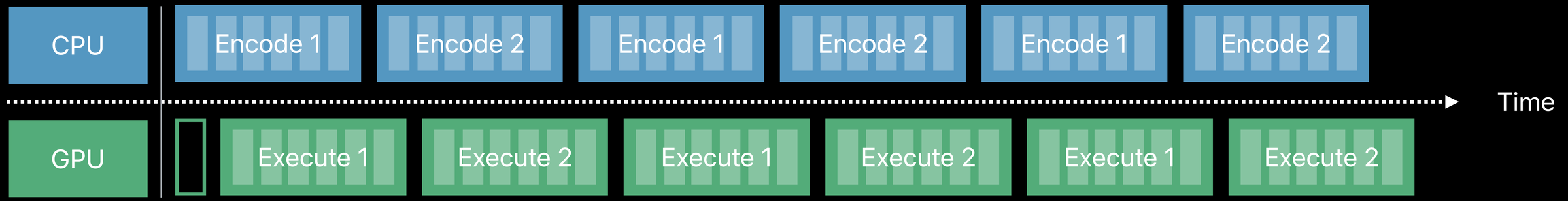

```
let cmdBuf = MPSCommandBuffer(commandQueue: queue)

let result = myGraph.encodeBatch(commandBuffer: cmdBuf, sourceImages: inputs)
```

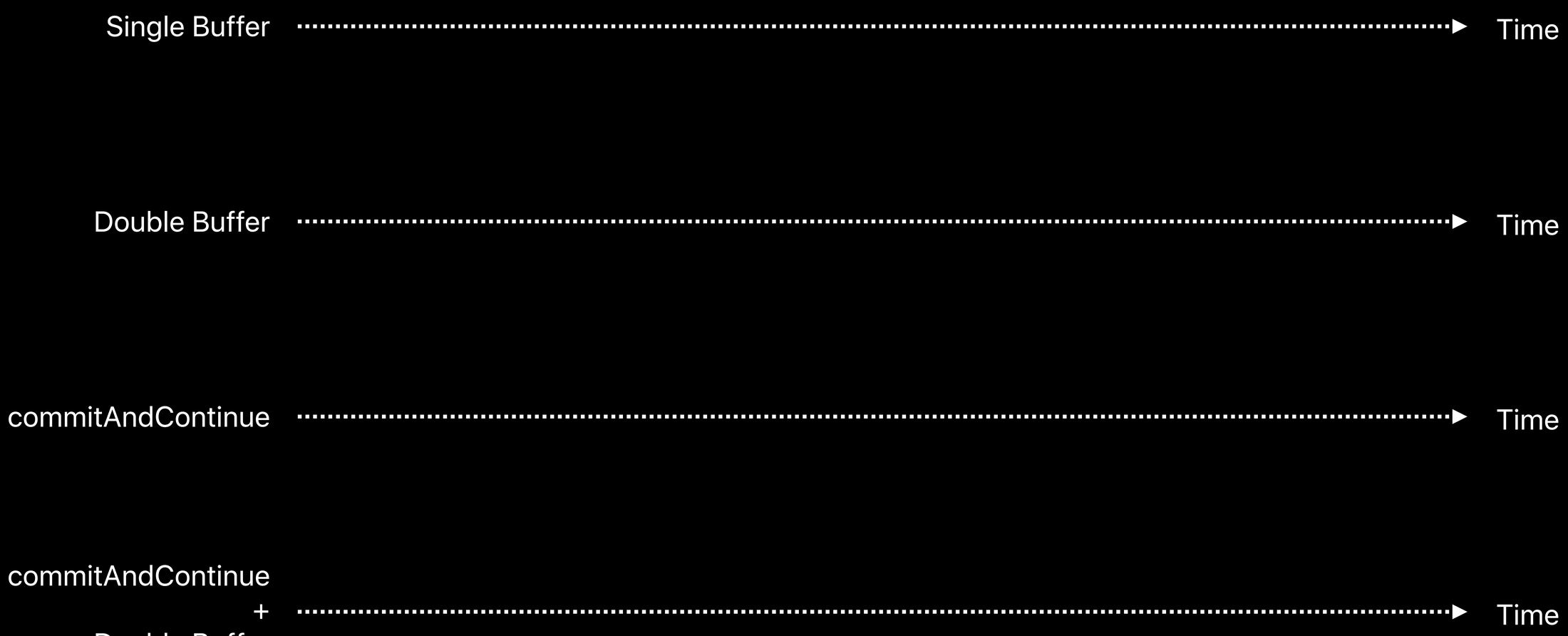
commitAndContinue

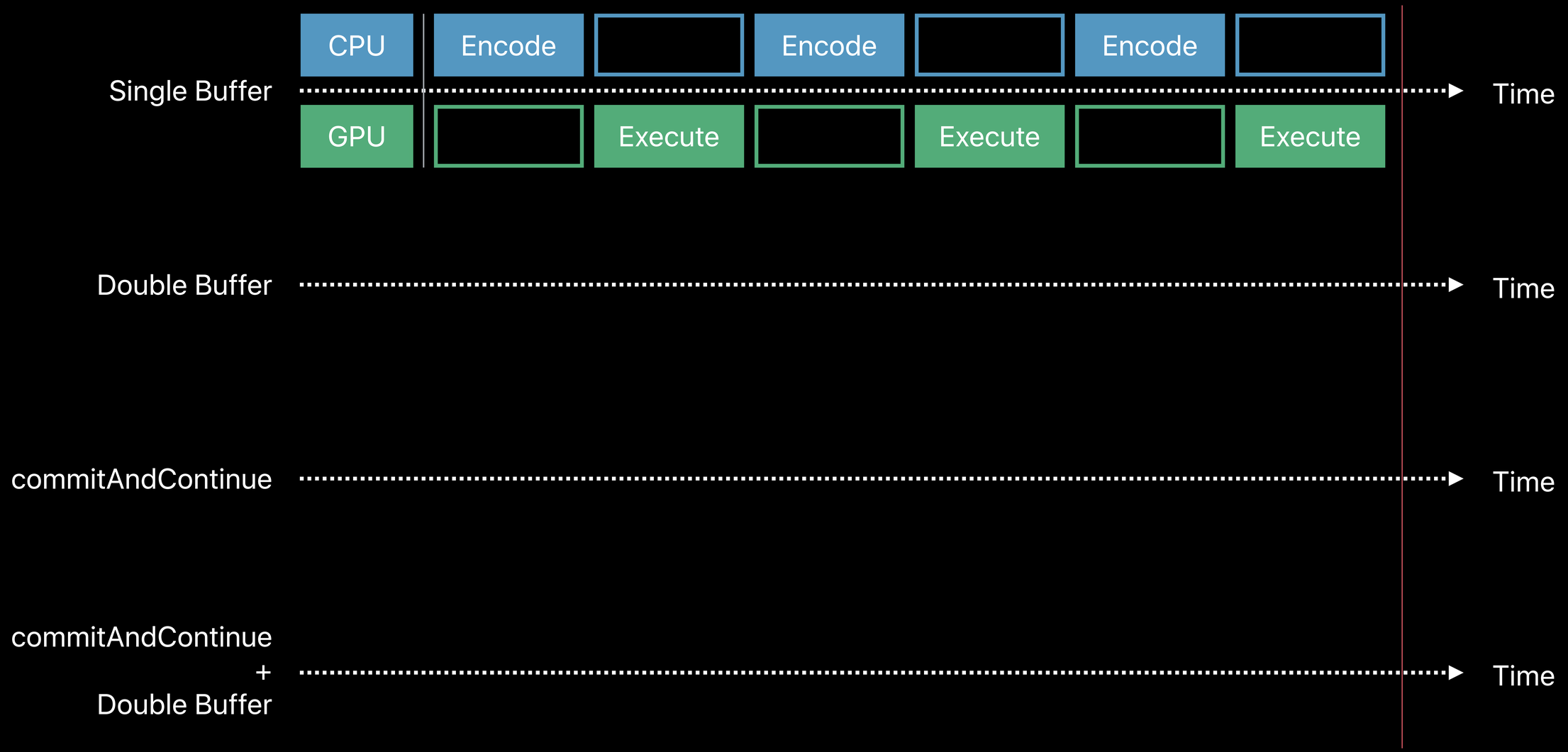


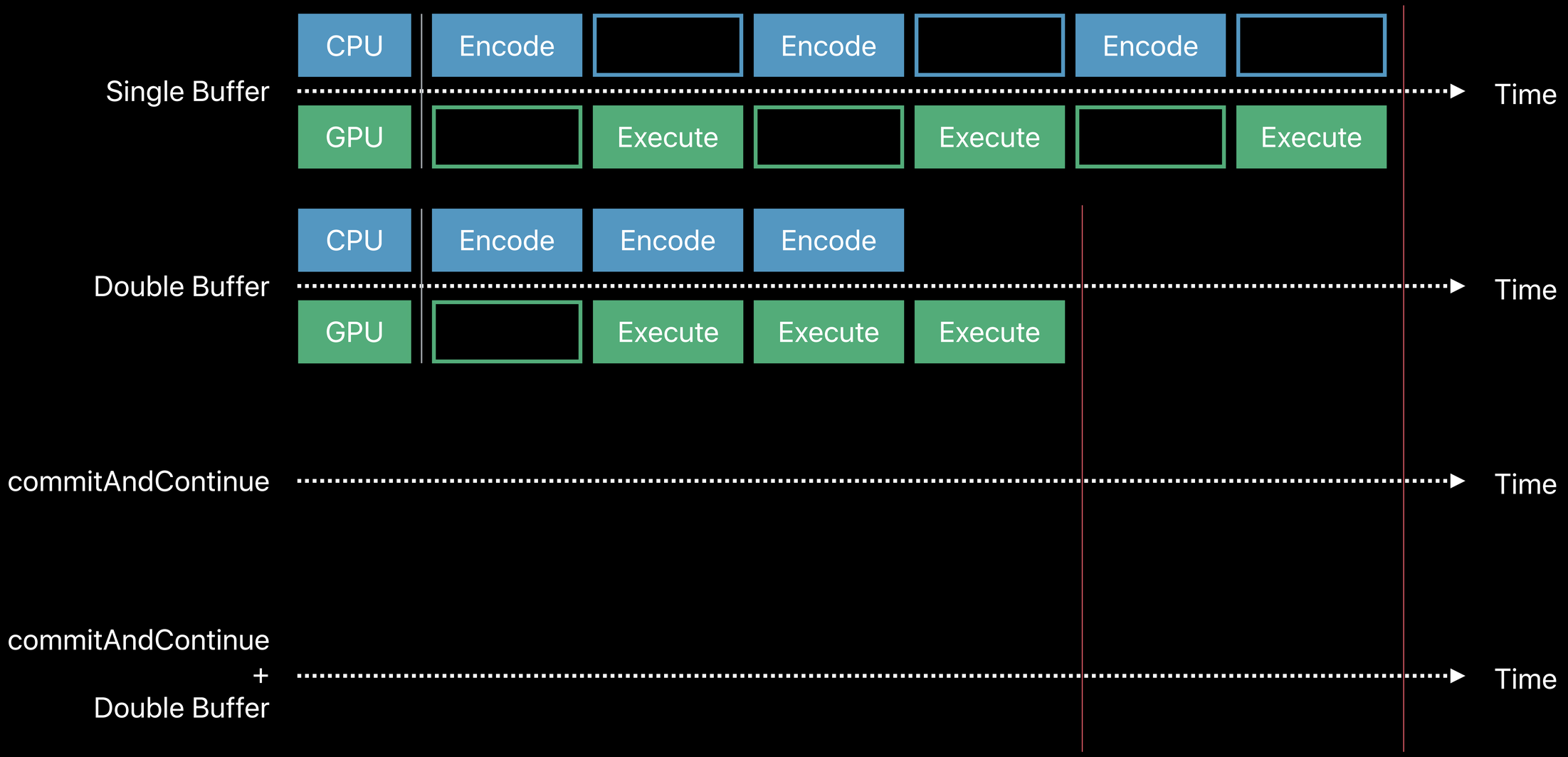

```
let cmdBuf = MPSCommandBuffer(commandQueue: queue)

let result = myGraph.encodeBatch(commandBuffer: cmdBuf, sourceImages: inputs)
```

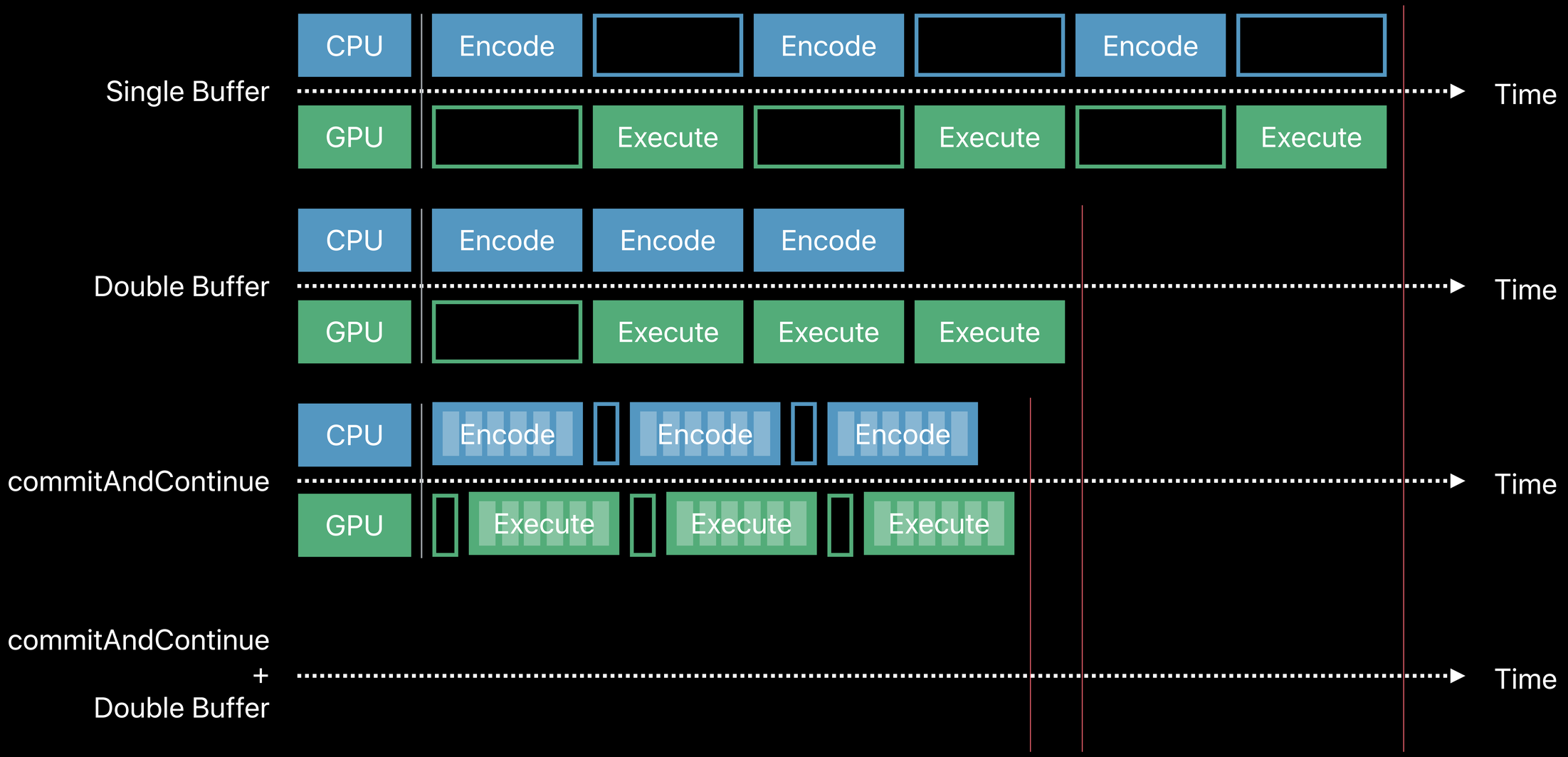
commitAndContinue with Double Buffering

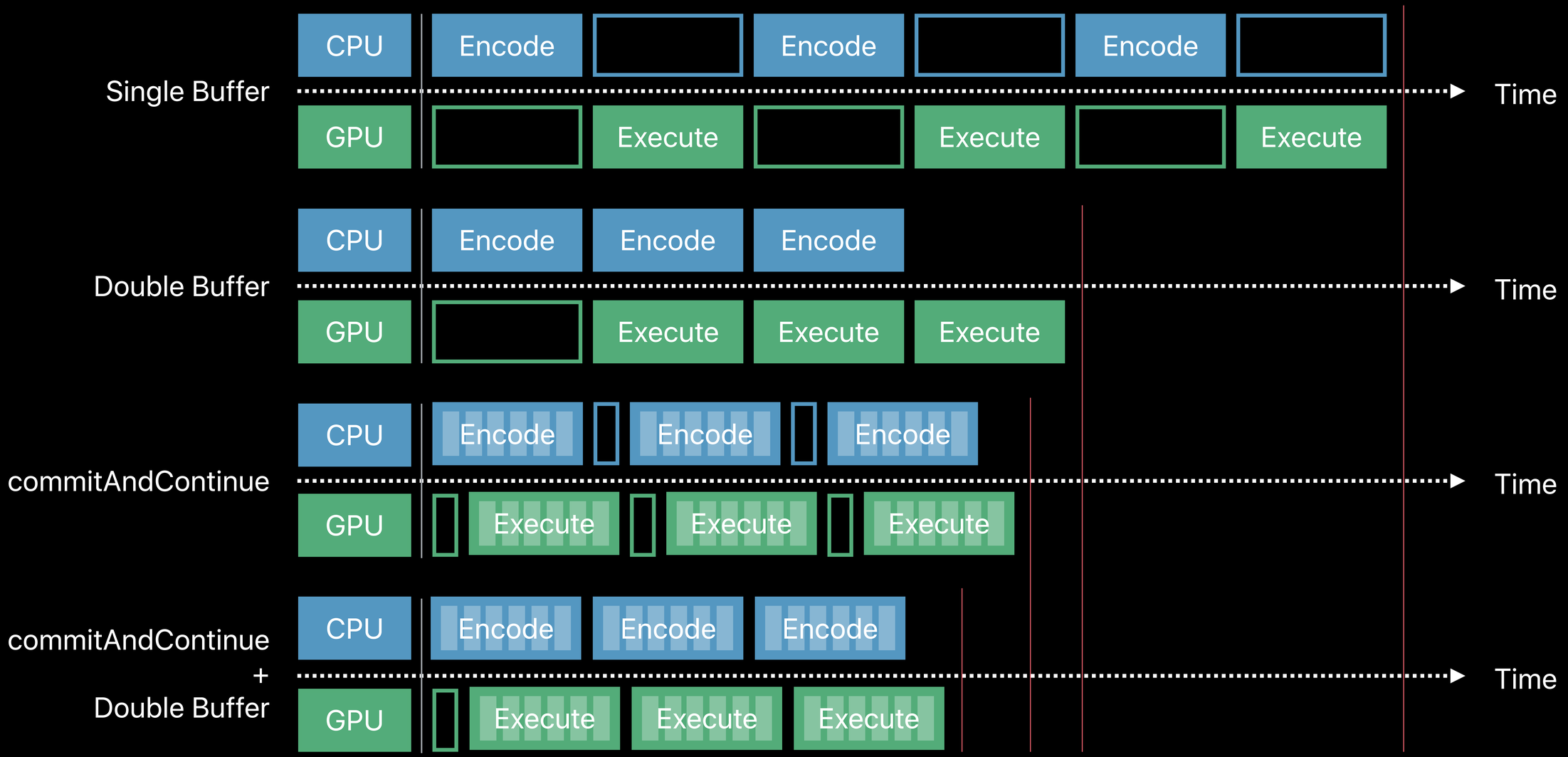



Single Buffer
Time
Double Buffer
Time
commitAndContinue
Time
commitAndContinue
+
Double Buffer
Time


CPU
Encode
Encode
Encode
Single Buffer
Time
GPU
Execute
Execute
Execute
Double Buffer
Time
commitAndContinue
Time
commitAndContinue
+
Double Buffer
Time

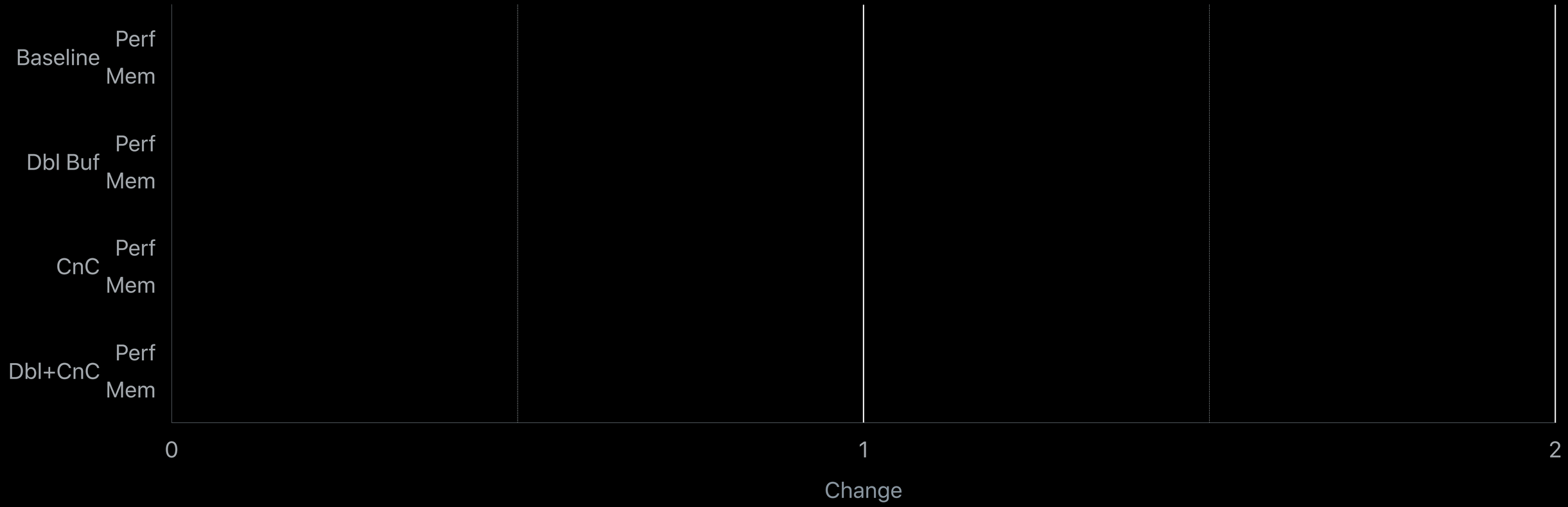

Single Buffer
CPU
Encode
Encode
Encode
GPU
Execute
Execute
Execute
Time
Double Buffer
CPU
Encode
Encode
Encode
GPU
Execute
Execute
Execute
Time
commitAndContinue
Time
commitAndContinue
+
Double Buffer
Time


Single Buffer
CPU
Encode
Encode
Encode
GPU
Execute
Execute
Execute
Time
Double Buffer
CPU
Encode
Encode
Encode
GPU
Execute
Execute
Execute
Time
commitAndContinue
CPU
Encode
Encode
Encode
GPU
Execute
Execute
Execute
Time
commitAndContinue
+
Double Buffer
Time


Single Buffer
CPU
Encode
Encode
Encode
GPU
Execute
Execute
Execute
Time
Double Buffer
CPU
Encode
Encode
Encode
GPU
Execute
Execute
Execute
Time
commitAndContinue
CPU
Encode
Encode
Encode
GPU
Execute
Execute
Execute
Time
commitAndContinue
+
Double Buffer
CPU
Encode
Encode
Encode
GPU
Execute
Execute
Execute
Time

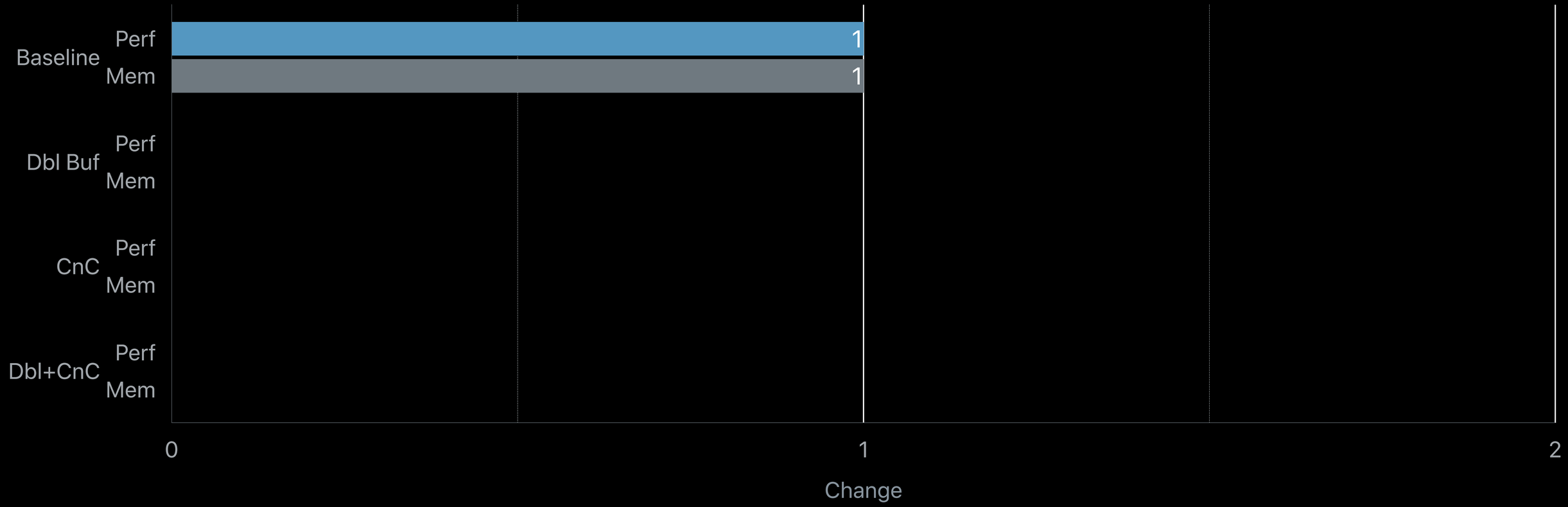
commitAndContinue

ResNet 50



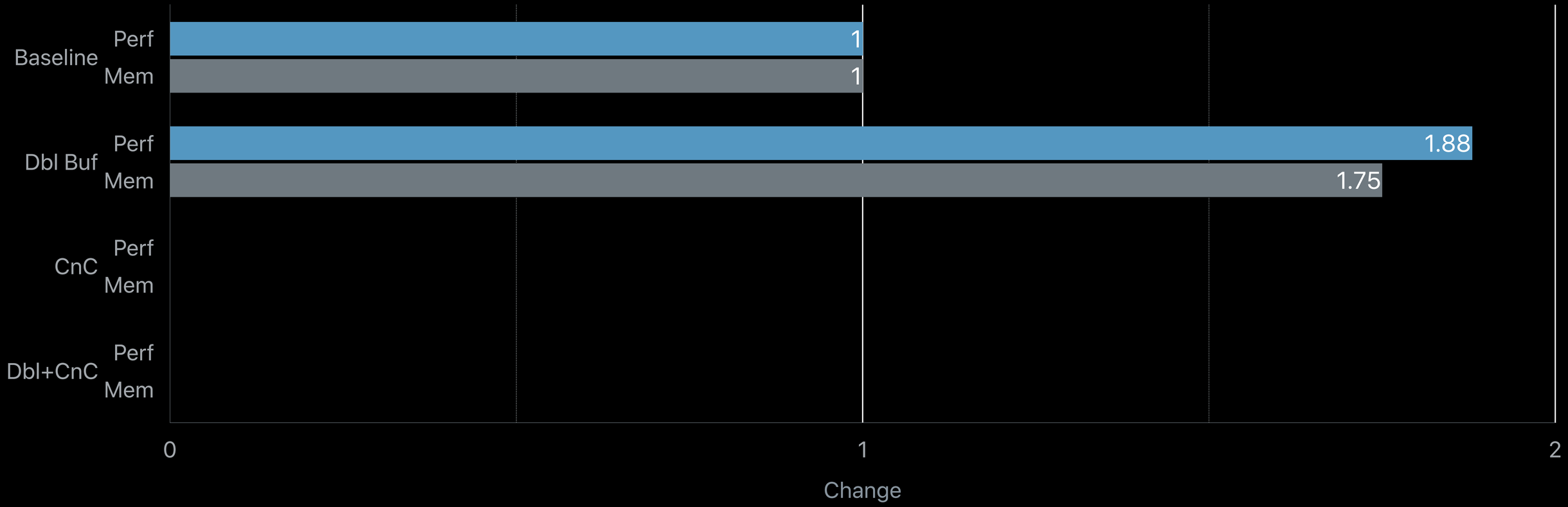
commitAndContinue

ResNet 50



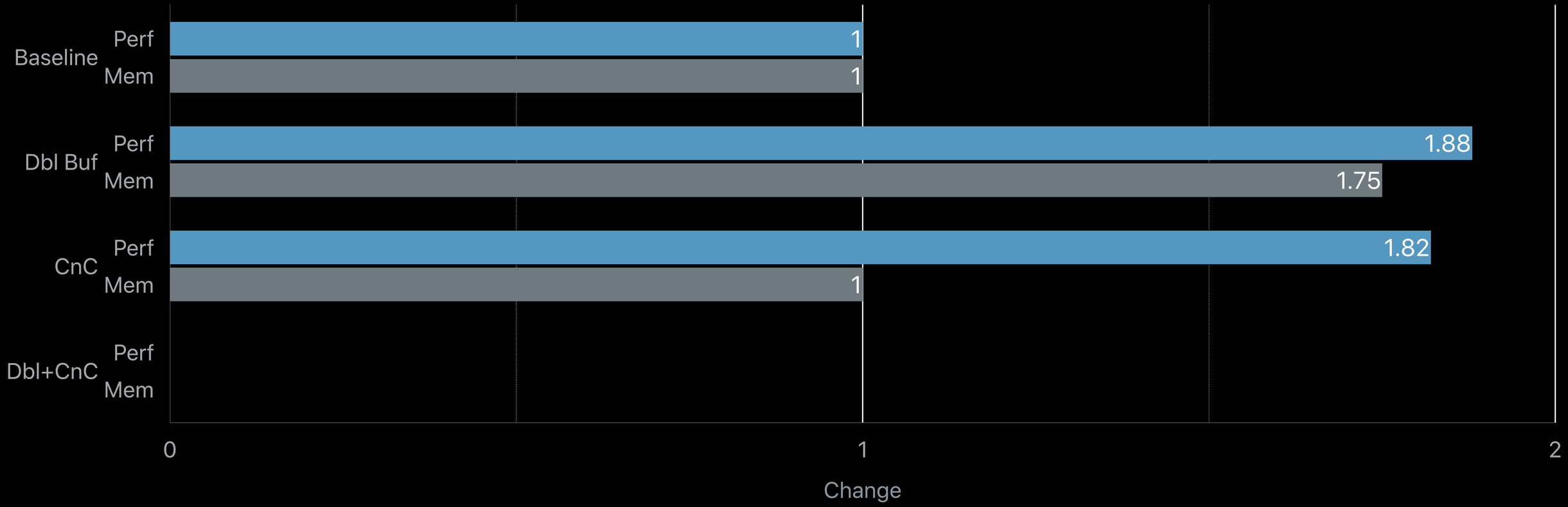
commitAndContinue

ResNet 50



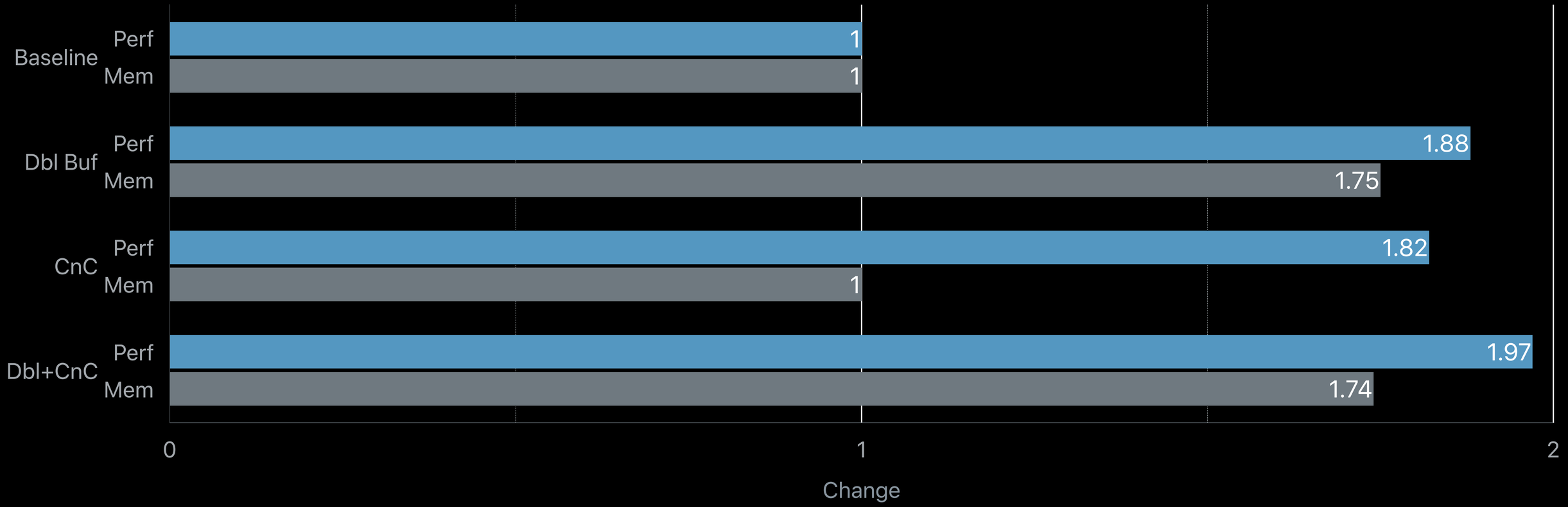
commitAndContinue

ResNet 50



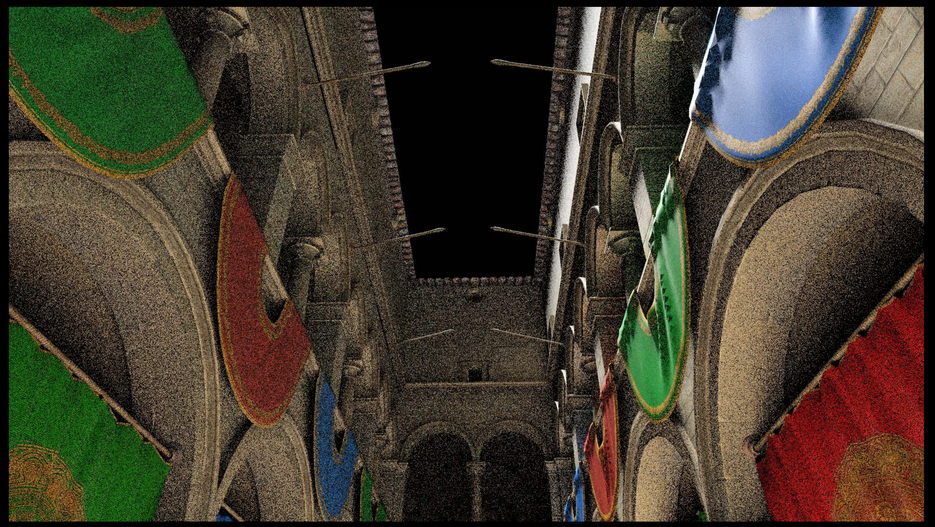
commitAndContinue

ResNet 50



ML Denoising

Example



Noisy

Clean

ML Denoising

Example

Create training graph

Run training

Deploy inference on new images

ML Denoising

Example

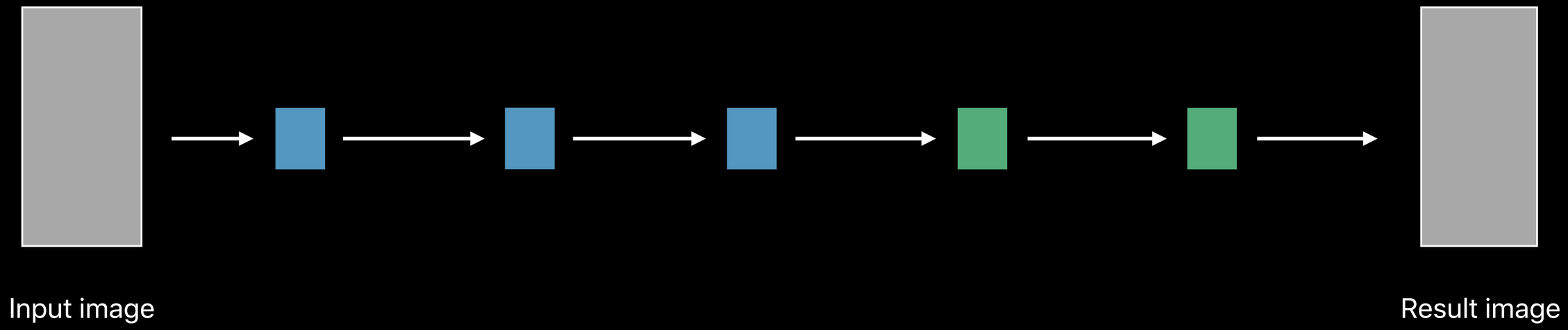
Create training graph

Run training

Deploy inference on new images

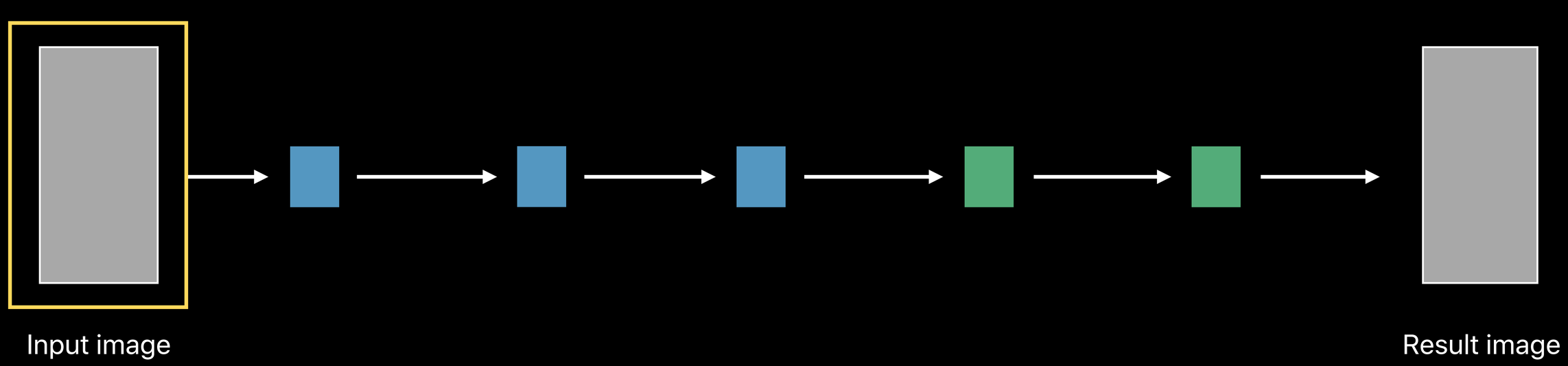
ML Denoising — Inference

Example



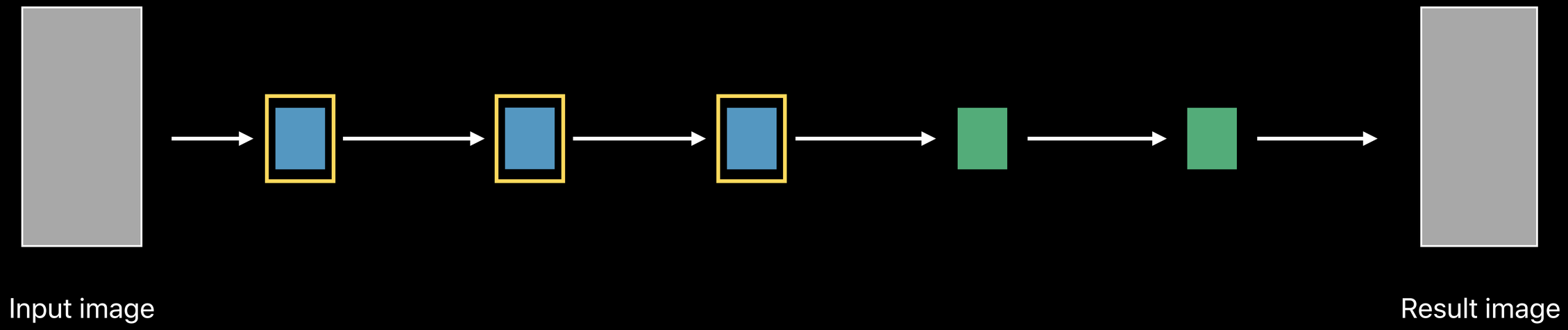
ML Denoising — Inference

Example



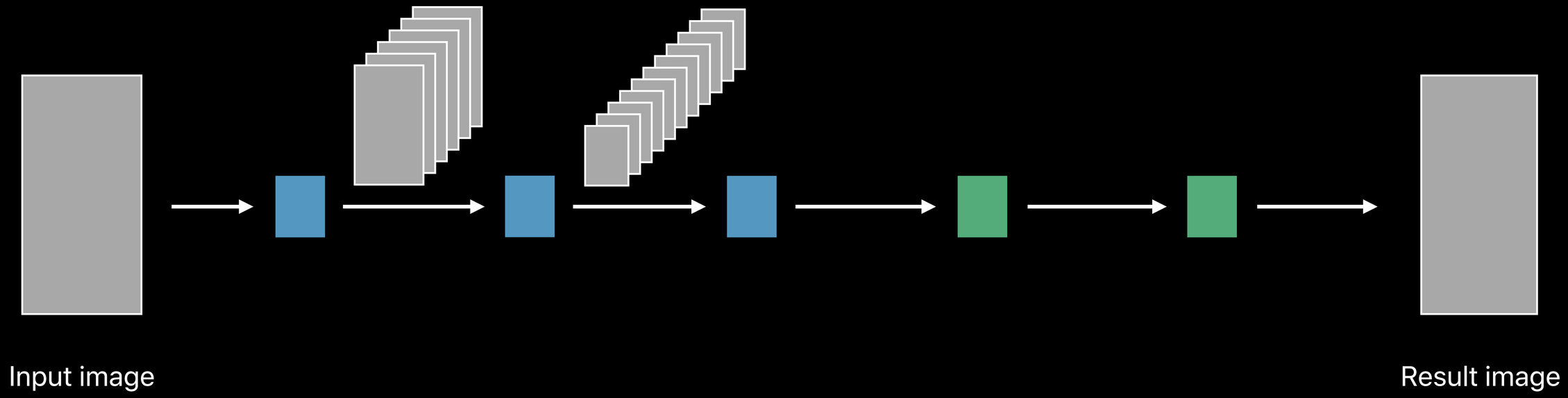
ML Denoising — Inference

Example



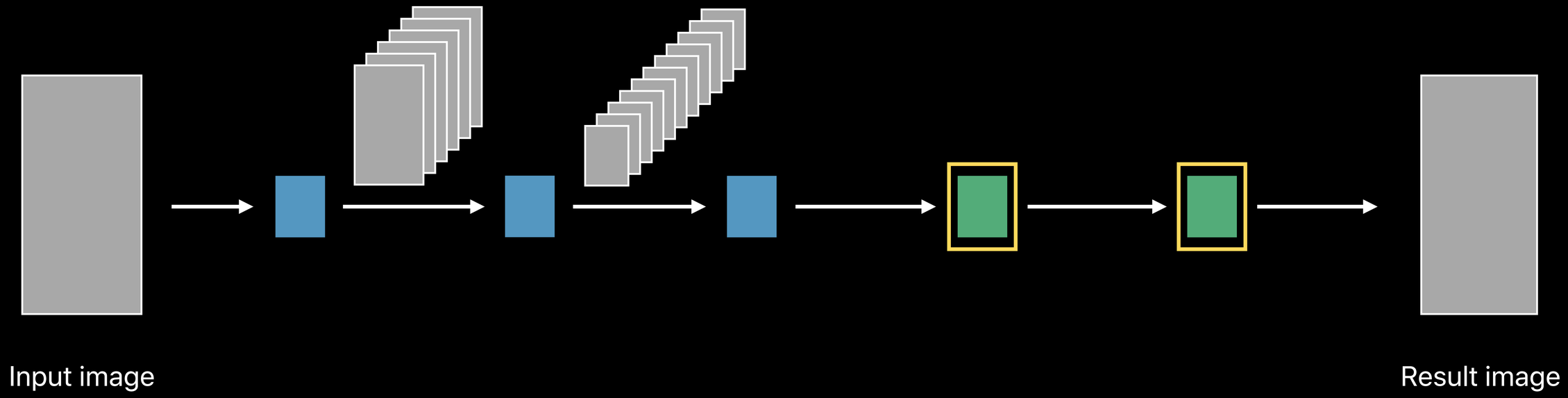
ML Denoising — Inference

Example



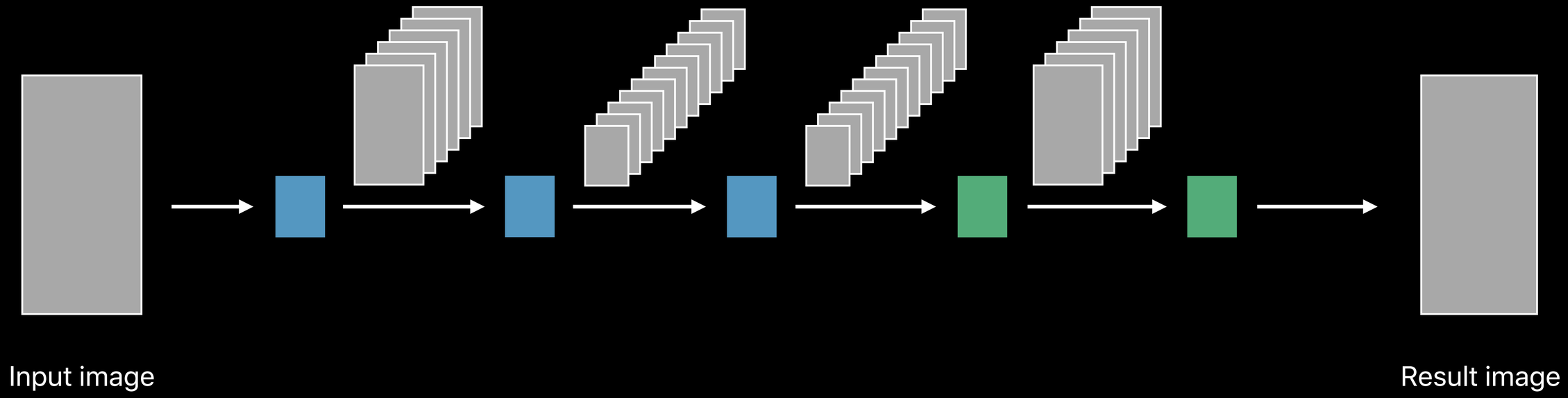
ML Denoising — Inference

Example



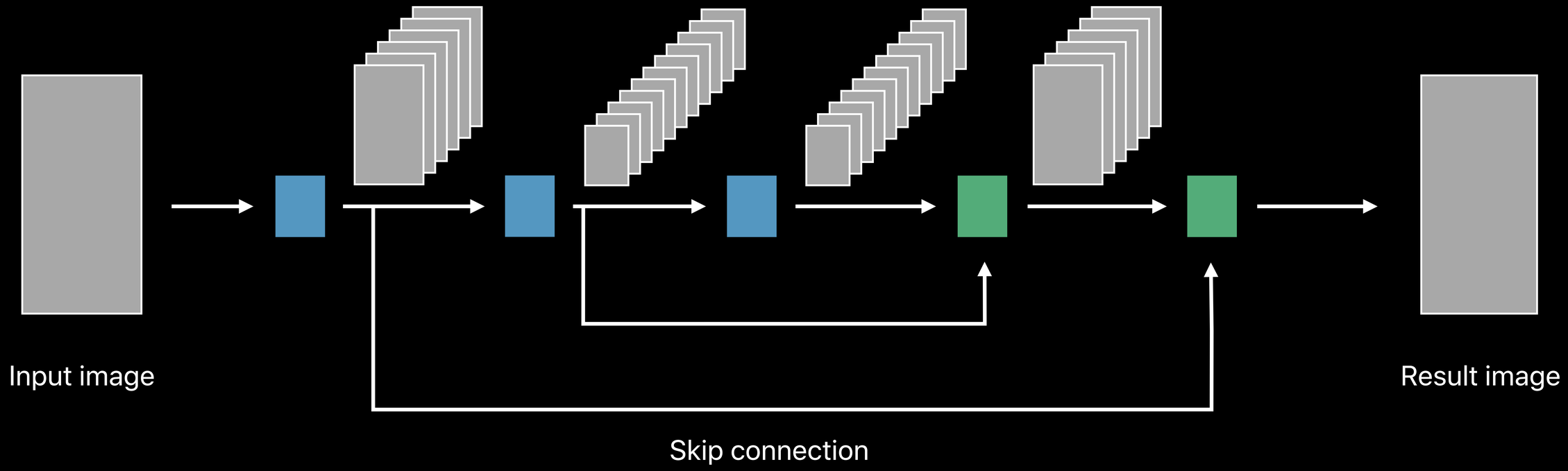
ML Denoising — Inference

Example



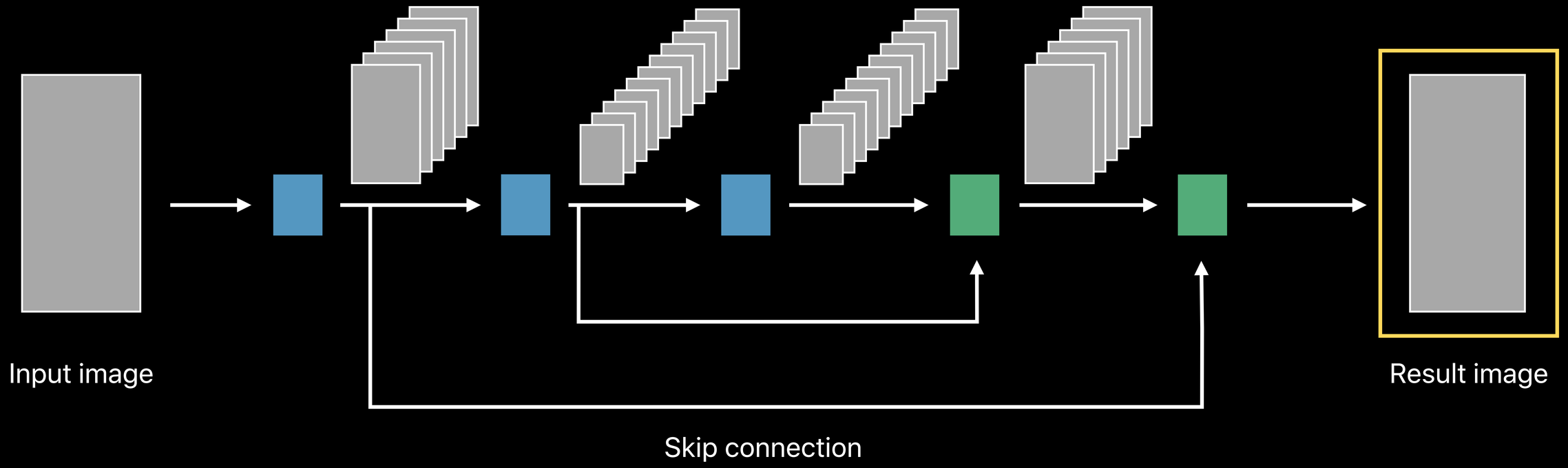
ML Denoising — Inference

Example



ML Denoising — Inference

Example



ML Denoising — Encoder Architecture

Example

Convolution

ReLu

Max Pooling

ML Denoising — Encoder Architecture

Example

Convolution

ReLu

Max Pooling

ML Denoising — Encoder Architecture

Example

Convolution

ReLu

Max Pooling

ML Denoising — Encoder Architecture

Example

Convolution

ReLu

Max Pooling

ML Denoising — Encoder Architecture

Example

Convolution

ReLu

Max Pooling

```
func encoderNode(input: MPSNNImageNode) -> MPSNNFilterNode{
   let conv1 = MPSCNNConvolutionNode(source:  input, weights: MyWeights(file:"conv1.dat"))
   let relu1 = MPSCNNNeuronReLUNode(source:   conv1.resultImage, a: 0.1)
   let conv2 = MPSCNNConvolutionNode(source:  relu1.resultImage, weights: MyWeights(file:"conv2.dat"))
   let relu2 = MPSCNNNeuronReLUNode(source:   conv2.resultImage, a: 0.1)
   let conv3 = MPSCNNConvolutionNode(source:  relu2.resultImage, weights: MyWeights(file:"conv3.dat"))
   let relu3 = MPSCNNNeuronReLUNode(source:   conv3.resultImage, a: 0.1)
   let pool4 = MPSCNNPoolingMaxNode(source:   relu3.resultImage, filterSize: 2)
   return pool4
}
```

ML Denoising — Encoder Architecture

Example

Convolution

ReLu

Max Pooling

```
func encoderNode(input: MPSNNImageNode) -> MPSNNFilterNode{
   let conv1 = MPSCNNConvolutionNode(source:  input, weights: MyWeights(file:"conv1.dat"))
   let relu1 = MPSCNNNeuronReLUNode(source:   conv1.resultImage, a: 0.1)
   let conv2 = MPSCNNConvolutionNode(source:  relu1.resultImage, weights: MyWeights(file:"conv2.dat"))
   let relu2 = MPSCNNNeuronReLUNode(source:   conv2.resultImage, a: 0.1)
   let conv3 = MPSCNNConvolutionNode(source:  relu2.resultImage, weights: MyWeights(file:"conv3.dat"))
   let relu3 = MPSCNNNeuronReLUNode(source:   conv3.resultImage, a: 0.1)
   let pool4 = MPSCNNPoolingMaxNode(source:   relu3.resultImage, filterSize: 2)
   return pool4
}
```

ML Denoising — Decoder Architecture

Example

Convolution

Upsampling

ReLu

Addition

ML Denoising — Decoder Architecture

Example

Convolution

Upsampling

ReLu

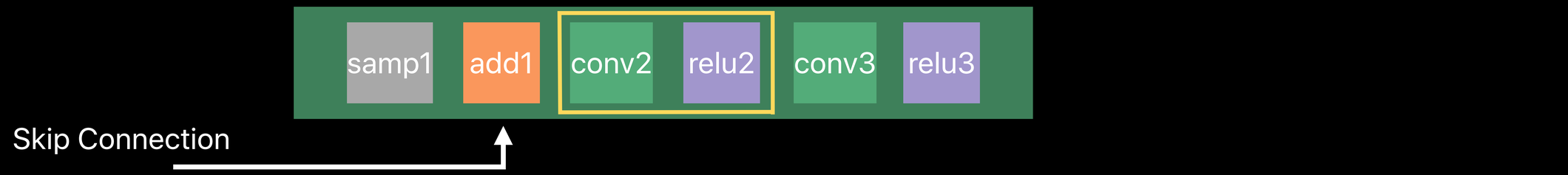
Addition

ML Denoising — Decoder Architecture

Example

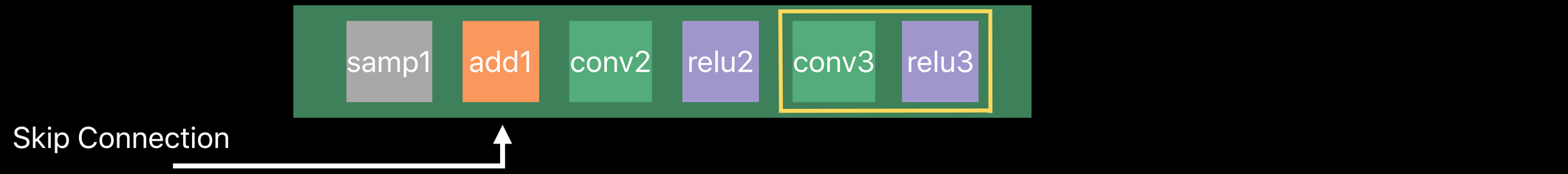
ML Denoising — Decoder Architecture

Example



ML Denoising — Decoder Architecture

Example



ML Denoising — Decoder Architecture

Example

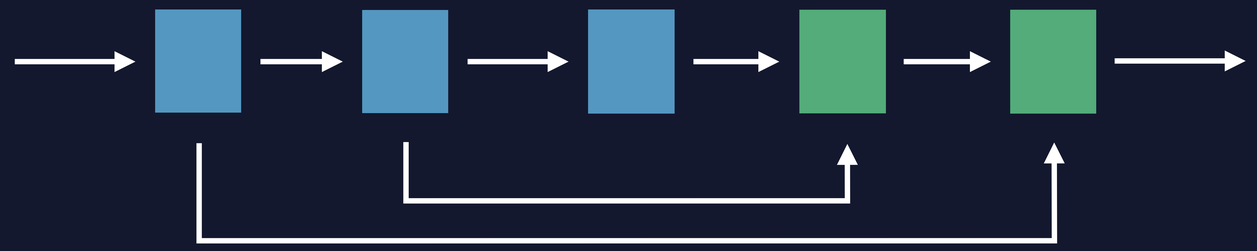

```
func decoderNode(input: MPSNNImageNode, skip: MPSNNImageNode) -> MPSNNFilterNode{
   let samp1 = MPSCNNUpsamplingNearestNode(source: input, integerScaleFactorX: 2, integerScaleFactorY: 2)
   let add1  = MPSNNAdditionNode(sources: [skip, samp1.resultImage])
   let conv2 = MPSCNNConvolutionNode(source:  add1.resultImage, weights: MyWeights(file:“conv2.dat”))
   let relu2 = MPSCNNNeuronReLUNode(source:   conv2.resultImage, a: 0.1)
   let conv3 = MPSCNNConvolutionNode(source:  relu2.resultImage, weights: MyWeights(file:“conv2.dat”))
   let relu3 = MPSCNNNeuronReLUNode(source:   conv3.resultImage, a: 0.1)
   return relu3
}
```

ML Denoising — Decoder Architecture

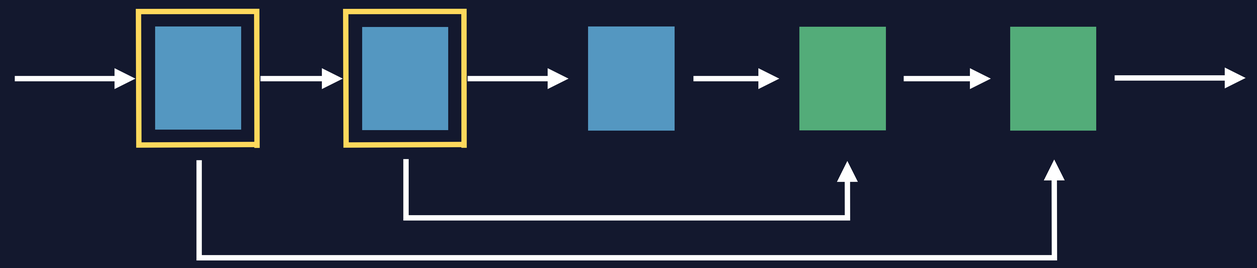
Example


```
func decoderNode(input: MPSNNImageNode, skip: MPSNNImageNode) -> MPSNNFilterNode{
   let samp1 = MPSCNNUpsamplingNearestNode(source: input, integerScaleFactorX: 2, integerScaleFactorY: 2)
   let add1  = MPSNNAdditionNode(sources: [skip, samp1.resultImage])
   let conv2 = MPSCNNConvolutionNode(source:  add1.resultImage, weights: MyWeights(file:“conv2.dat”))
   let relu2 = MPSCNNNeuronReLUNode(source:   conv2.resultImage, a: 0.1)
   let conv3 = MPSCNNConvolutionNode(source:  relu2.resultImage, weights: MyWeights(file:“conv2.dat”))
   let relu3 = MPSCNNNeuronReLUNode(source:   conv3.resultImage, a: 0.1)
   return relu3
}
```

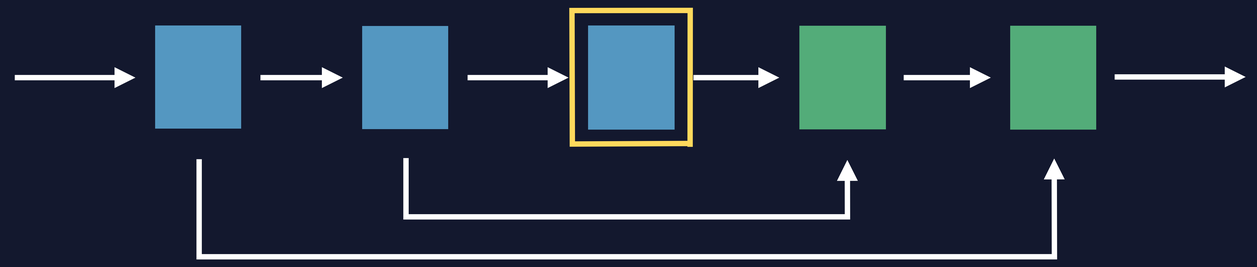
```
func inferenceGraph() -> MPSNNImageNode
{
    // Encoder nodes.
    let enc0 = encoderNode(MPSNNImageNode(handle:nil))
    let enc1 = encoderNode(enc0.resultImage)

    // Bottleneck nodes.
    let bottleneck = bottleneckNode(enc1.resultImage)

    // Decoder nodes
    let dec0 = decoderNode(bottleneck.resultImage, enc1.resultImage)
    let dec1 = decoderNode(dec0.resultImage, enc0.resultImage)

    return dec1.resultImage
}
```



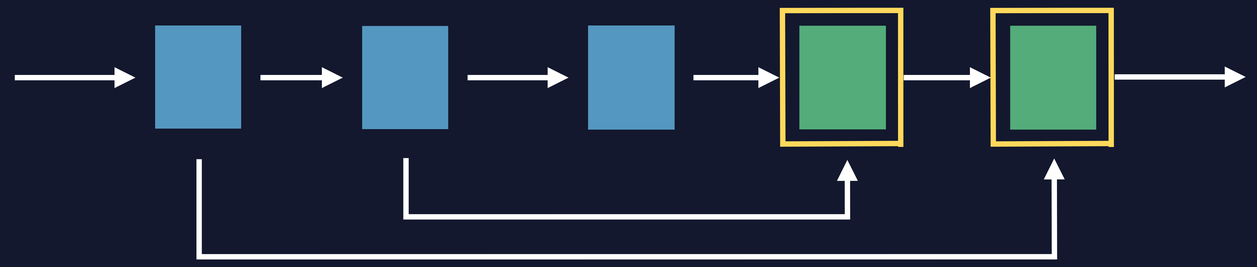
```
func inferenceGraph() -> MPSNNImageNode
{
    // Encoder nodes.
    let enc0 = encoderNode(MPSNNImageNode(handle:nil))
    let enc1 = encoderNode(enc0.resultImage)

    // Bottleneck nodes.
    let bottleneck = bottleneckNode(enc1.resultImage)

    // Decoder nodes
    let dec0 = decoderNode(bottleneck.resultImage, enc1.resultImage)
    let dec1 = decoderNode(dec0.resultImage, enc0.resultImage)

    return dec1.resultImage
}
```



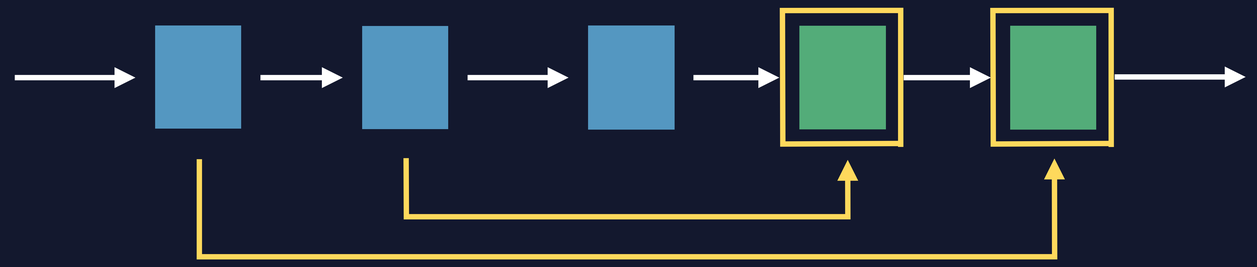
```
func inferenceGraph() -> MPSNNImageNode
{
    // Encoder nodes.
    let enc0 = encoderNode(MPSNNImageNode(handle:nil))
    let enc1 = encoderNode(enc0.resultImage)

    // Bottleneck nodes.
    let bottleneck = bottleneckNode(enc1.resultImage)

    // Decoder nodes
    let dec0 = decoderNode(bottleneck.resultImage, enc1.resultImage)
    let dec1 = decoderNode(dec0.resultImage, enc0.resultImage)

    return dec1.resultImage
}
```



```
func inferenceGraph() -> MPSNNImageNode
{
    // Encoder nodes.
    let enc0 = encoderNode(MPSNNImageNode(handle:nil))
    let enc1 = encoderNode(enc0.resultImage)

    // Bottleneck nodes.
    let bottleneck = bottleneckNode(enc1.resultImage)

    // Decoder nodes
    let dec0 = decoderNode(bottleneck.resultImage, enc1.resultImage)
    let dec1 = decoderNode(dec0.resultImage, enc0.resultImage)

    return dec1.resultImage
}
```

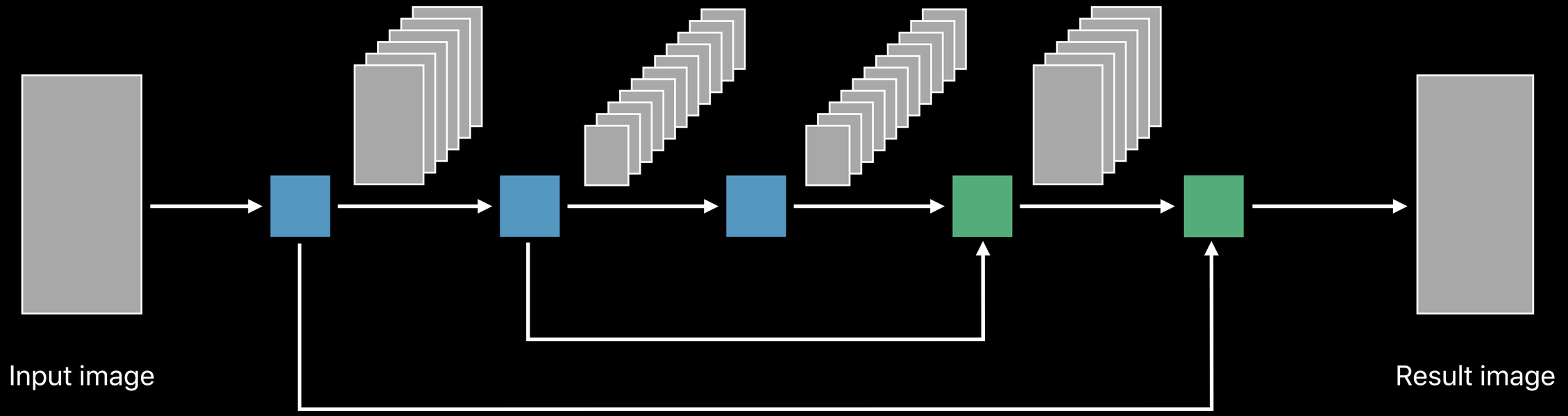


```
func inferenceGraph() -> MPSNNImageNode
{
    // Encoder nodes.
    let enc0 = encoderNode(MPSNNImageNode(handle:nil))
    let enc1 = encoderNode(enc0.resultImage)

    // Bottleneck nodes.
    let bottleneck = bottleneckNode(enc1.resultImage)

    // Decoder nodes
    let dec0 = decoderNode(bottleneck.resultImage, enc1.resultImage)
    let dec1 = decoderNode(dec0.resultImage, enc0.resultImage)

    return dec1.resultImage
}
```



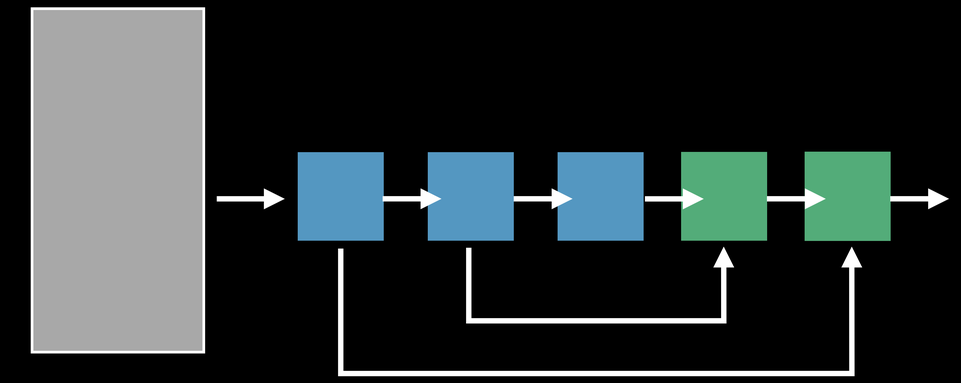
ML Denoising — Inference

Example



ML Denoising — Training

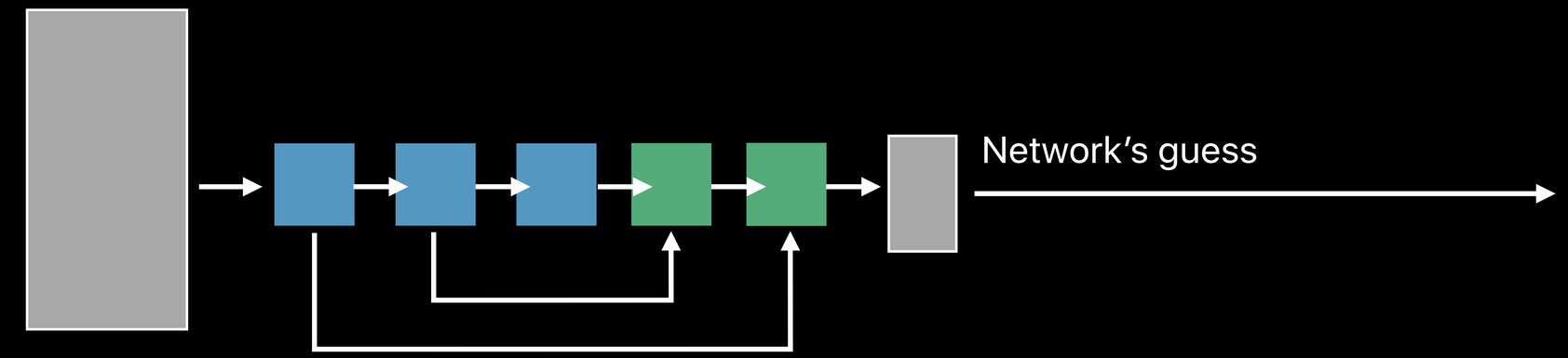
Example



Input image

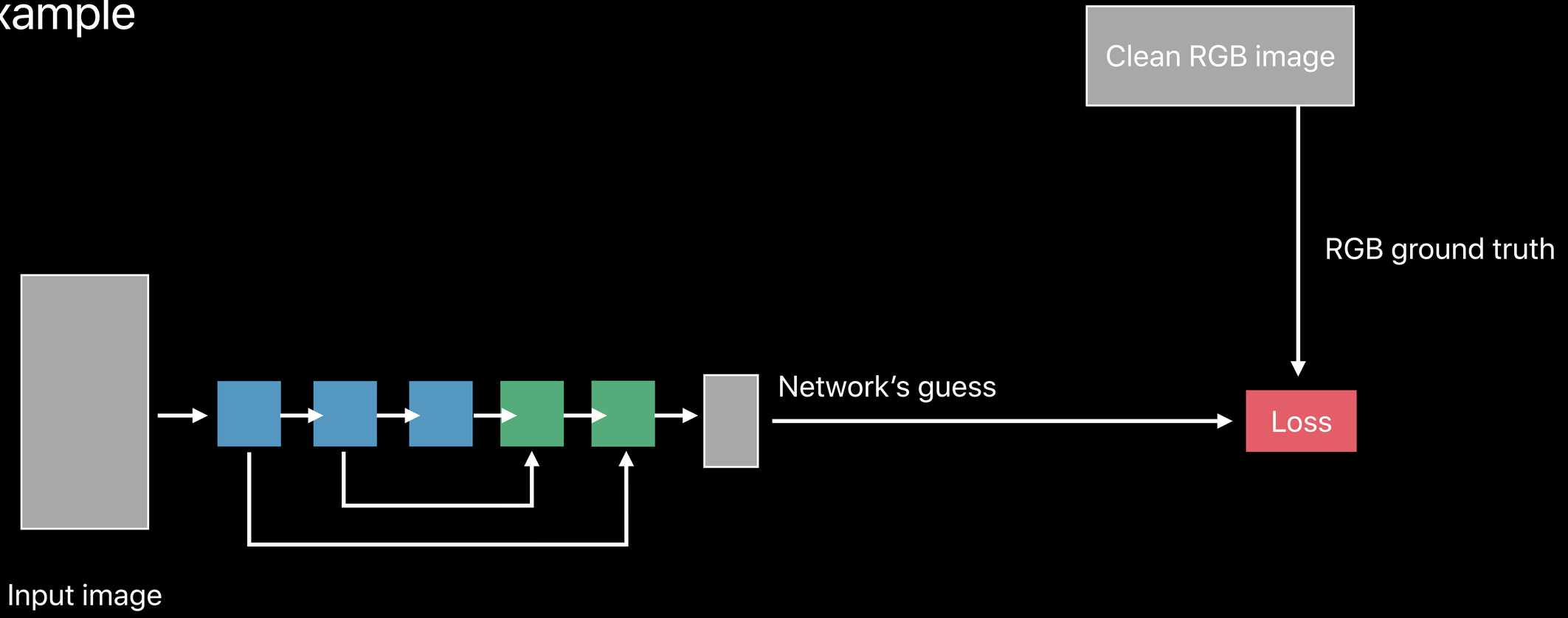
ML Denoising — Training

Example



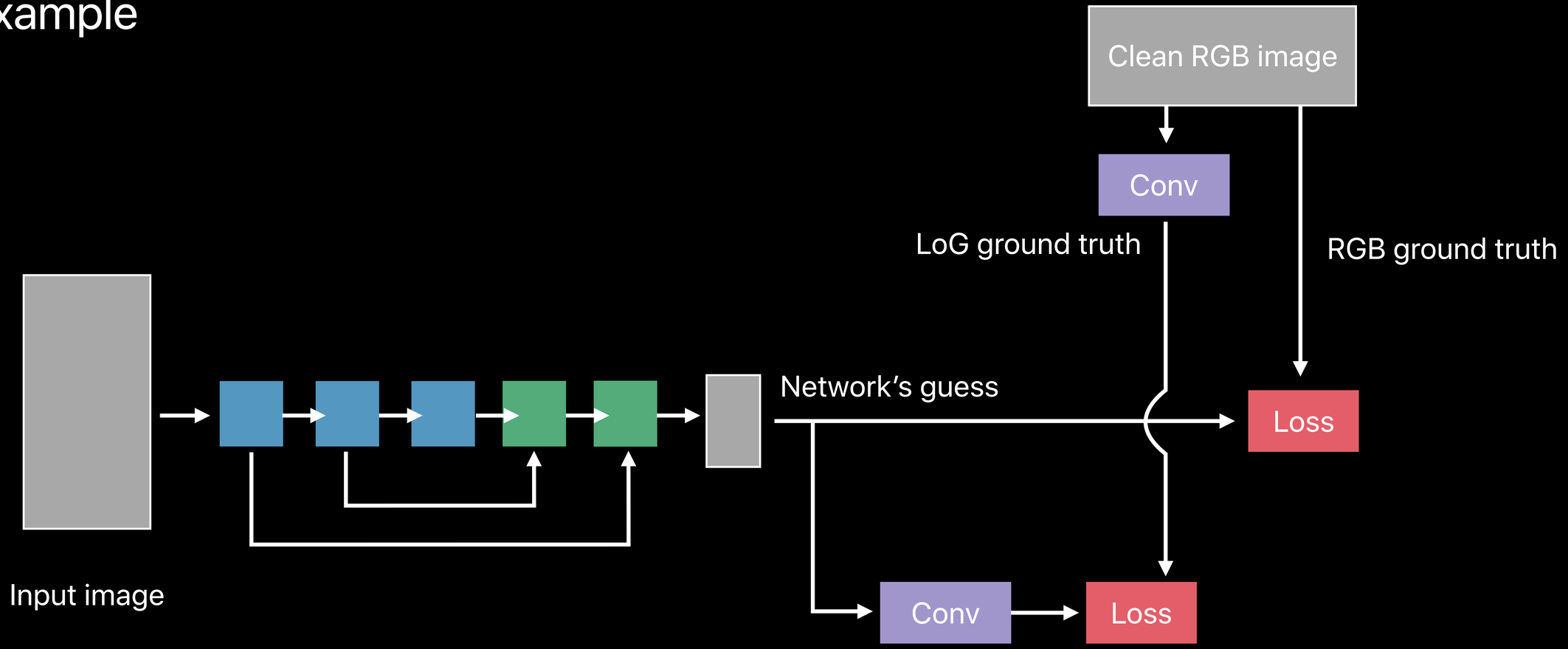
ML Denoising — Training

Example



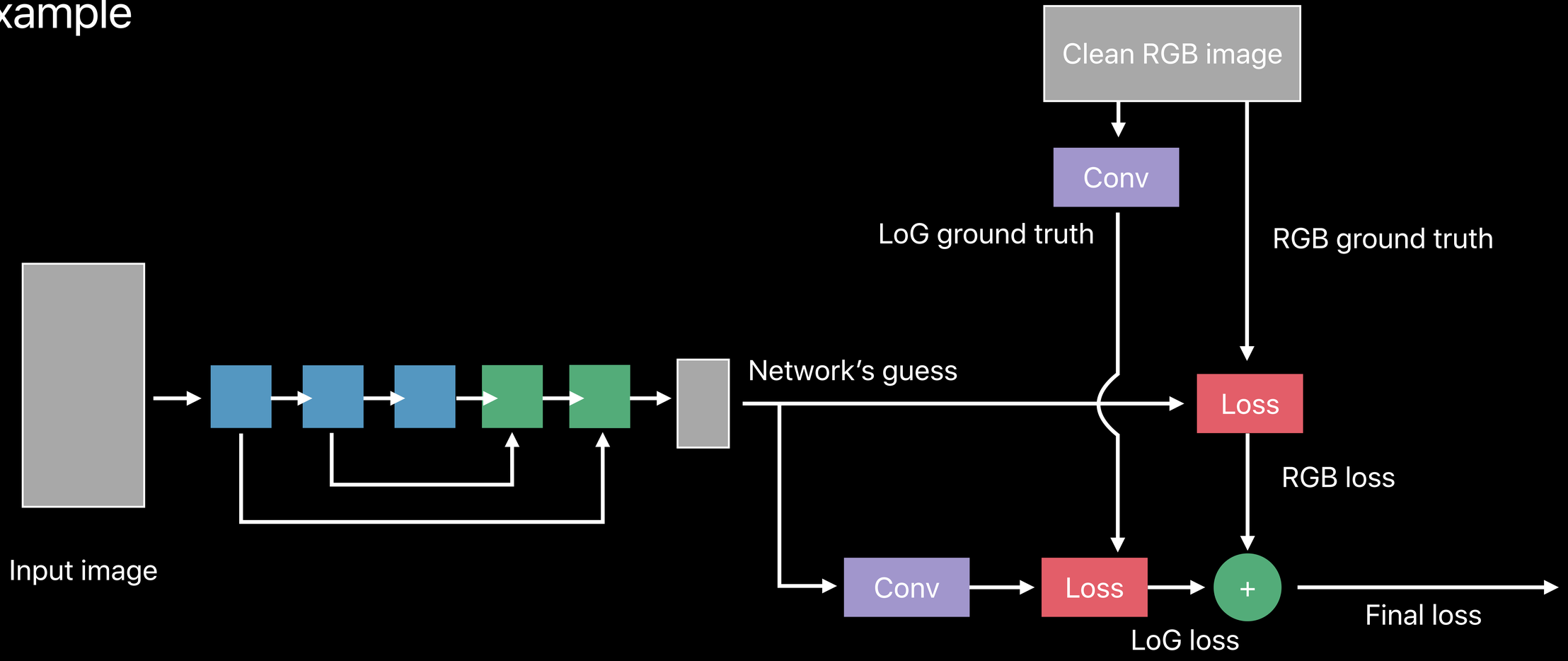
ML Denoising — Training

Example



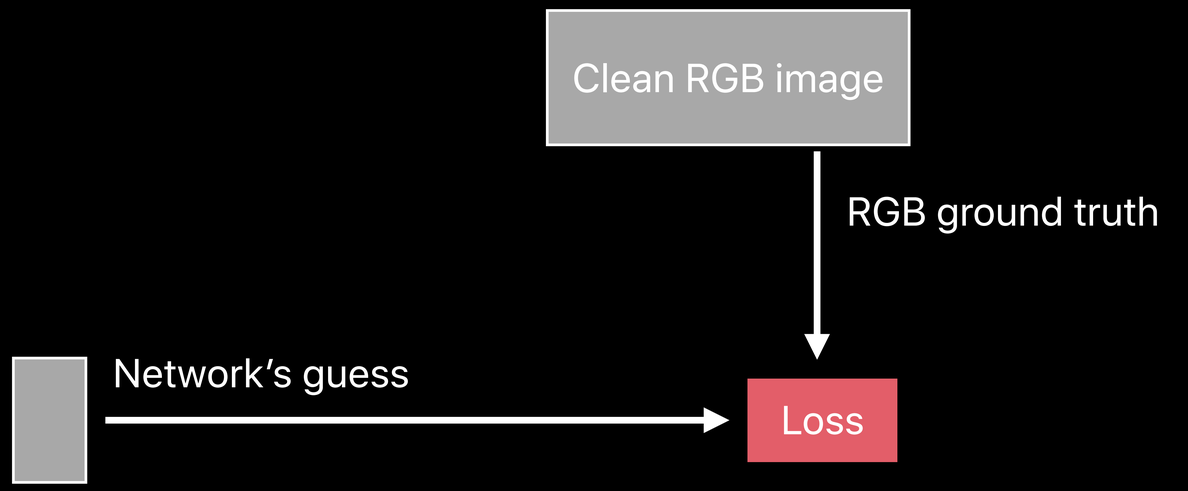
ML Denoising — Training

Example



ML Denoising

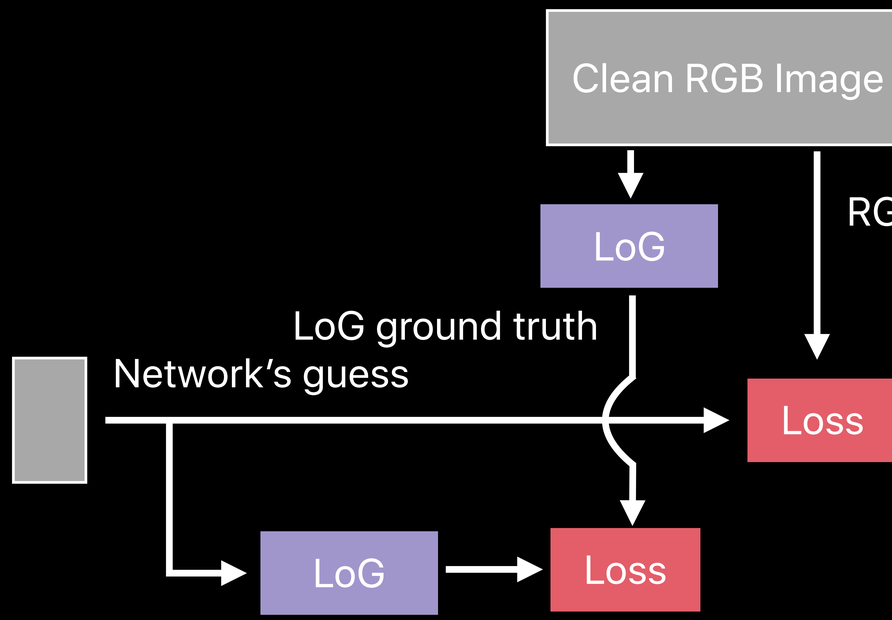
Example




```
let rgbLoss = MPSNNForwardLossNode(source:inferenceGraph().resultImage,
                                   labels:rgbImageNode.resultImage,
                                   lossDescriptor:lossDescriptor)
```

ML Denoising

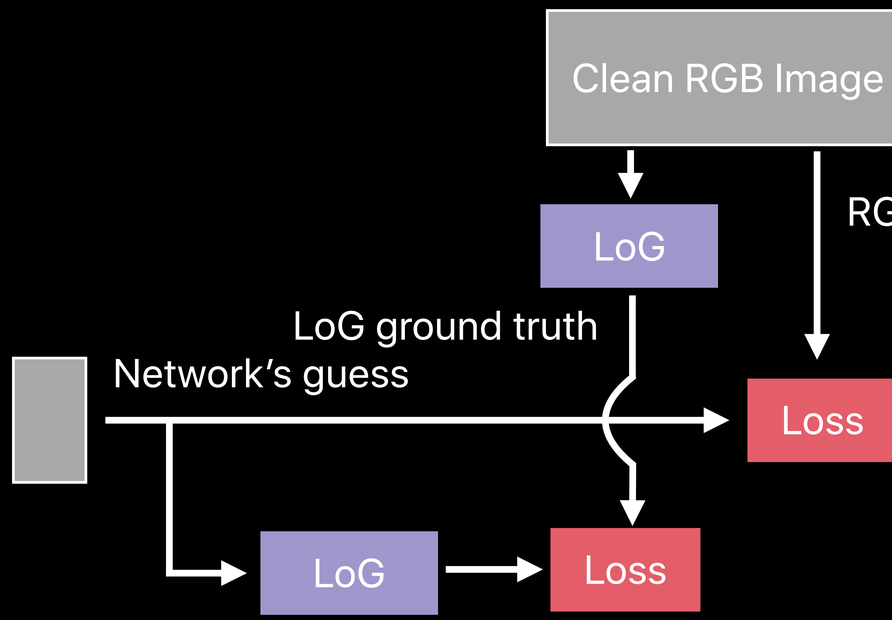
Example




```
let LoGNode0 = MPSCNNConvolutionNode(source:inferenceGraph().resultImage, weights: weights)
let LoGNode1 = MPSCNNConvolutionNode(source:rgbImageNode.resultImage, weights: weights)

let LoGLoss = MPSNNForwardLossNode(source:LoGNode0.resultImage,
                                   labels:LoGNode1.resultImage,
                                   lossDescriptor:desc)
let finalLoss = MPSNNAdditionNode(sources:[rgbLoss.resultImage, LoGLoss.resultImage])
```

ML Denoising

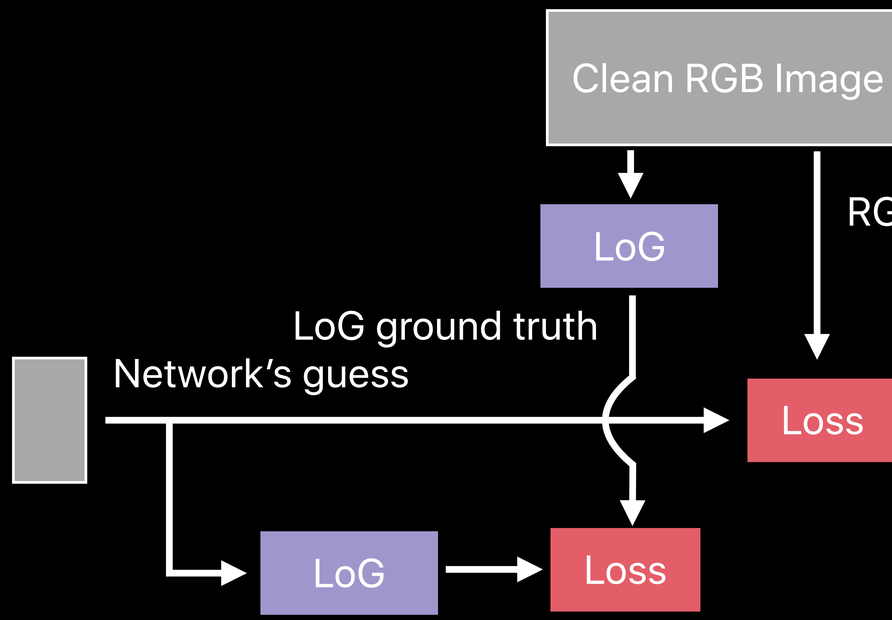
Example




```
let LoGNode0 = MPSCNNConvolutionNode(source:inferenceGraph().resultImage, weights: weights)
let LoGNode1 = MPSCNNConvolutionNode(source:rgbImageNode.resultImage, weights: weights)

let LoGLoss = MPSNNForwardLossNode(source:LoGNode0.resultImage,
                                   labels:LoGNode1.resultImage,
                                   lossDescriptor:desc)
let finalLoss = MPSNNAdditionNode(sources:[rgbLoss.resultImage, LoGLoss.resultImage])
```

ML Denoising

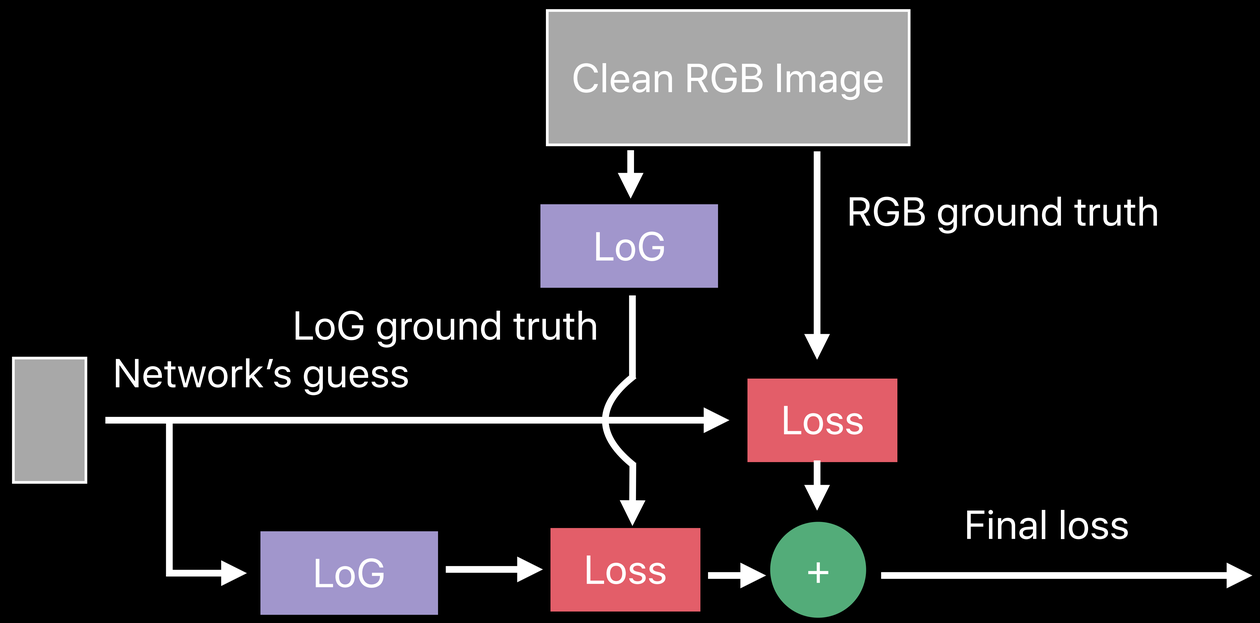
Example




```
let LoGNode0 = MPSCNNConvolutionNode(source:inferenceGraph().resultImage, weights: weights)
let LoGNode1 = MPSCNNConvolutionNode(source:rgbImageNode.resultImage, weights: weights)

let LoGLoss = MPSNNForwardLossNode(source:LoGNode0.resultImage,
                                   labels:LoGNode1.resultImage,
                                   lossDescriptor:desc)
let finalLoss = MPSNNAdditionNode(sources:[rgbLoss.resultImage, LoGLoss.resultImage])
```

ML Denoising

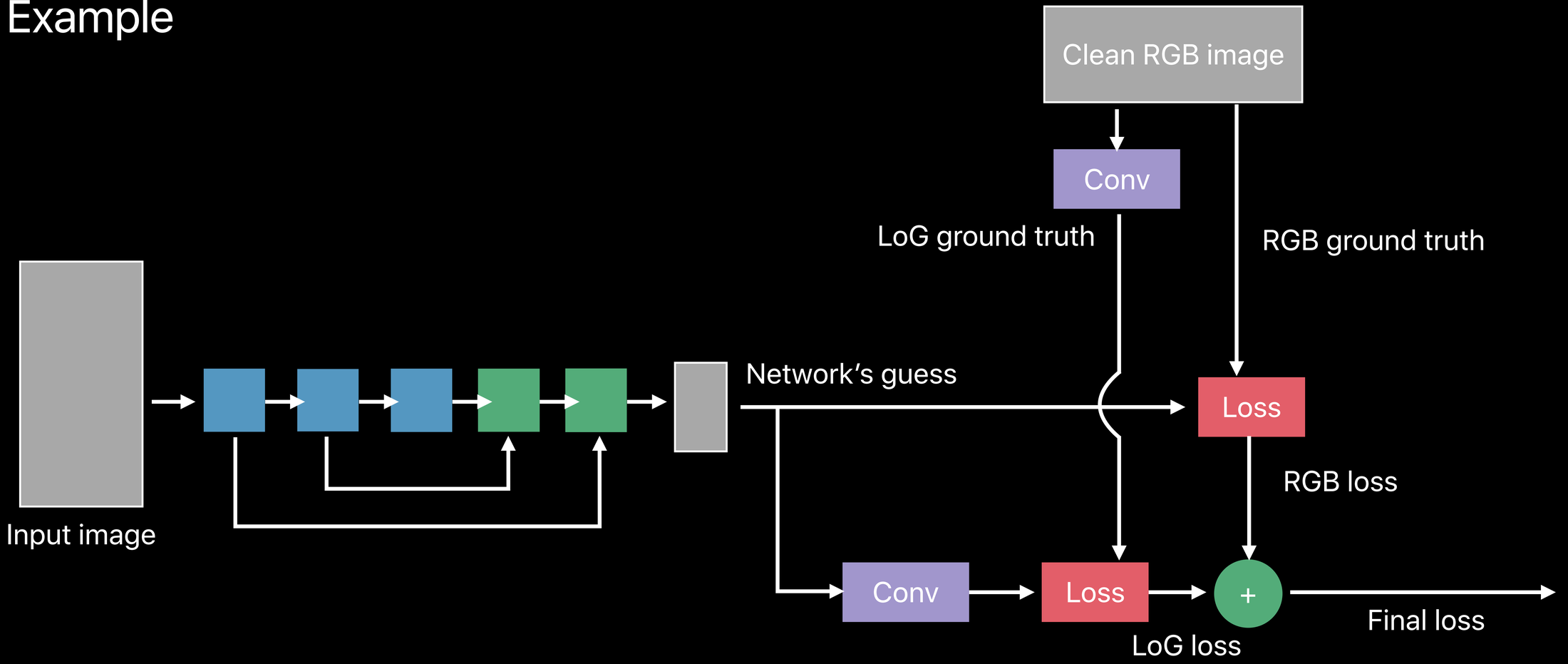
Example




```
let LoGNode0 = MPSCNNConvolutionNode(source:inferenceGraph().resultImage, weights: weights)
let LoGNode1 = MPSCNNConvolutionNode(source:rgbImageNode.resultImage, weights: weights)

let LoGLoss = MPSNNForwardLossNode(source:LoGNode0.resultImage,
                                   labels:LoGNode1.resultImage,
                                   lossDescriptor:desc)
let finalLoss = MPSNNAdditionNode(sources:[rgbLoss.resultImage, LoGLoss.resultImage])
```

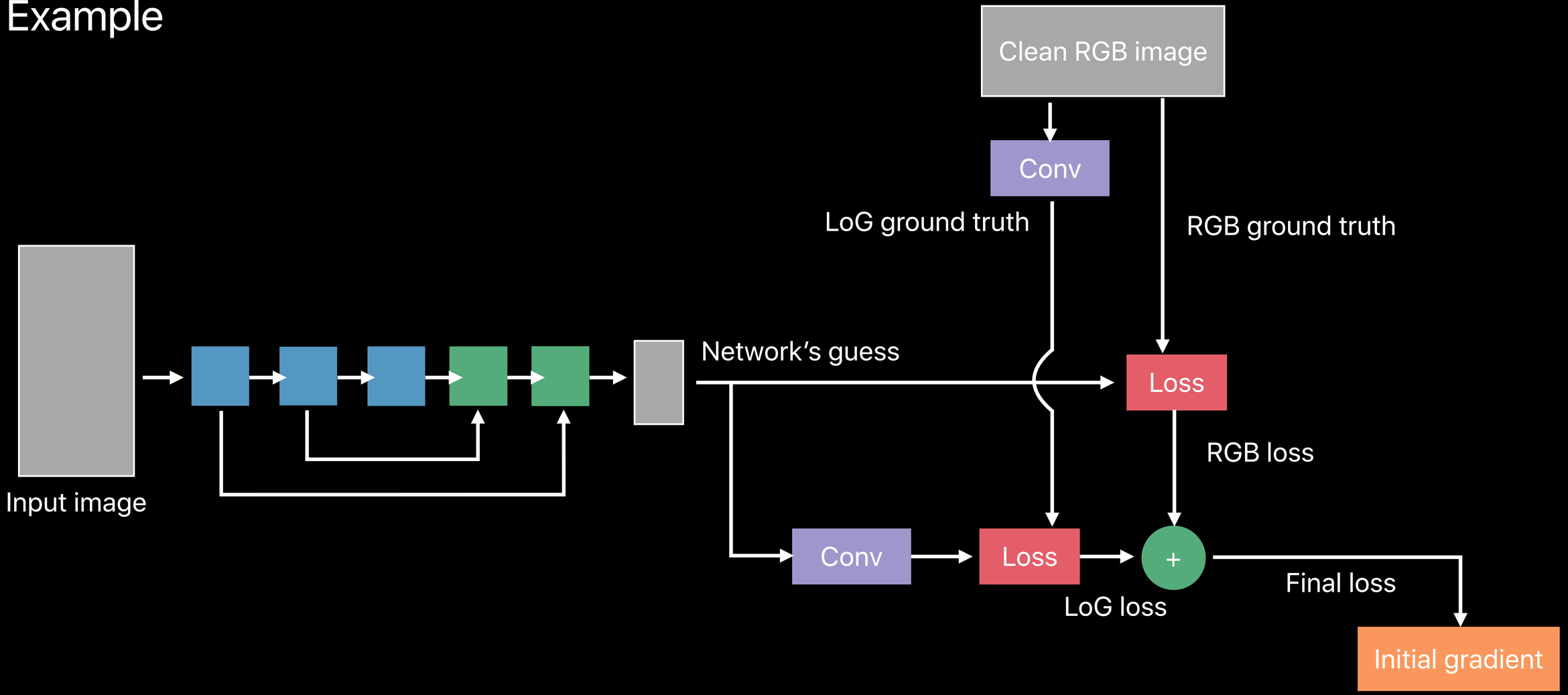
ML Denoising — Training

Example



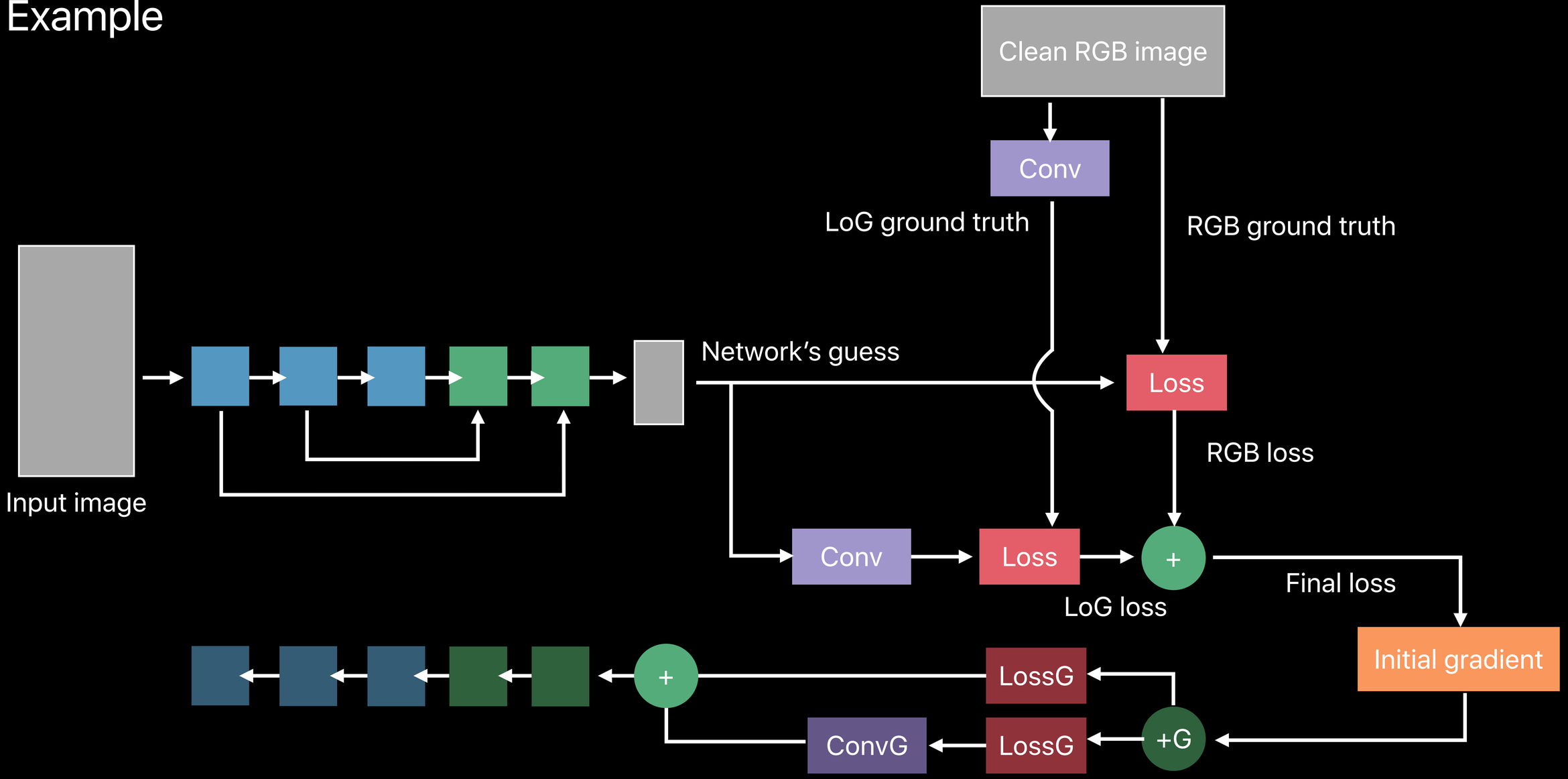
ML Denoising — Training

Example



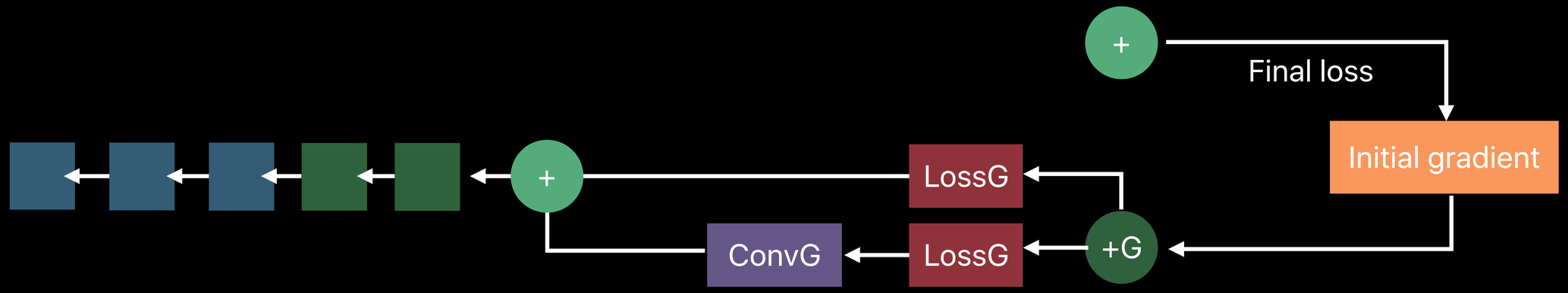
ML Denoising — Training

Example



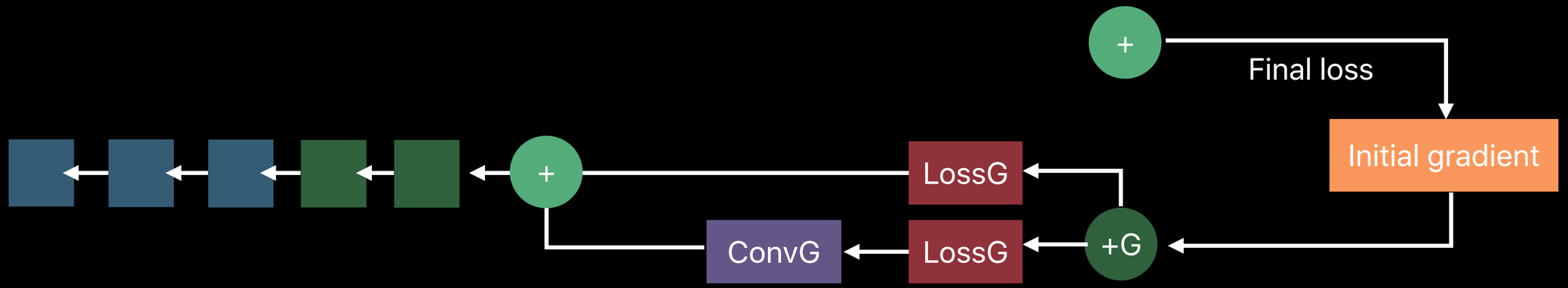
ML Denoising — Training

Example



ML Denoising

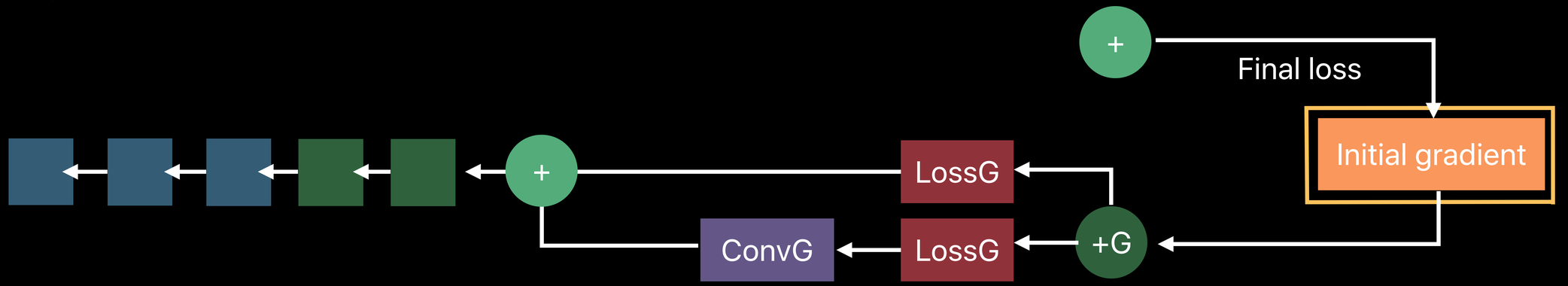
Example




```
let initGrad = MPSNNInitialGradientNode(source:finalLoss.resultImage)

let gradNodes = initGrad.trainingGraph(withSourceGradient:nil, nodeHandler:nil)
```

ML Denoising

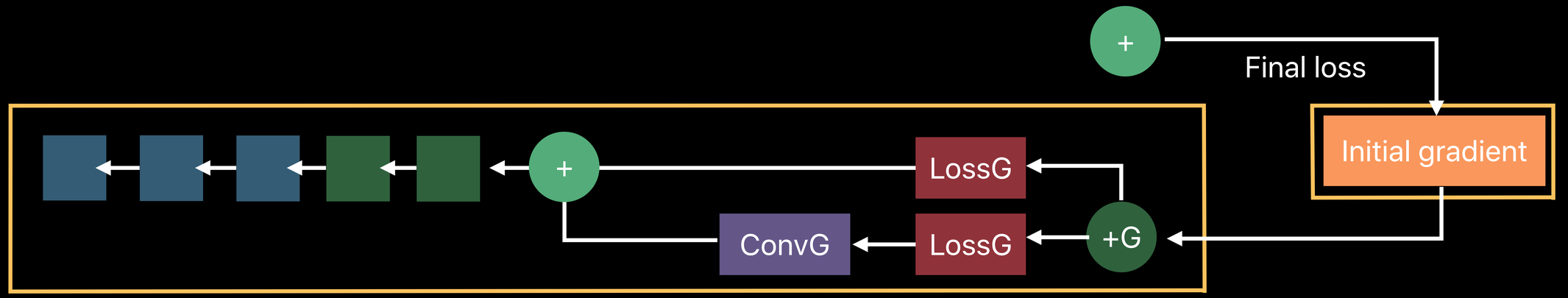
Example




```
let initGrad = MPSNNInitialGradientNode(source:finalLoss.resultImage)

let gradNodes = initGrad.trainingGraph(withSourceGradient:nil, nodeHandler:nil)
```

ML Denoising

Example




```
let initGrad = MPSNNInitialGradientNode(source:finalLoss.resultImage)

let gradNodes = initGrad.trainingGraph(withSourceGradient:nil, nodeHandler:nil)
```

ML Denoising

Example

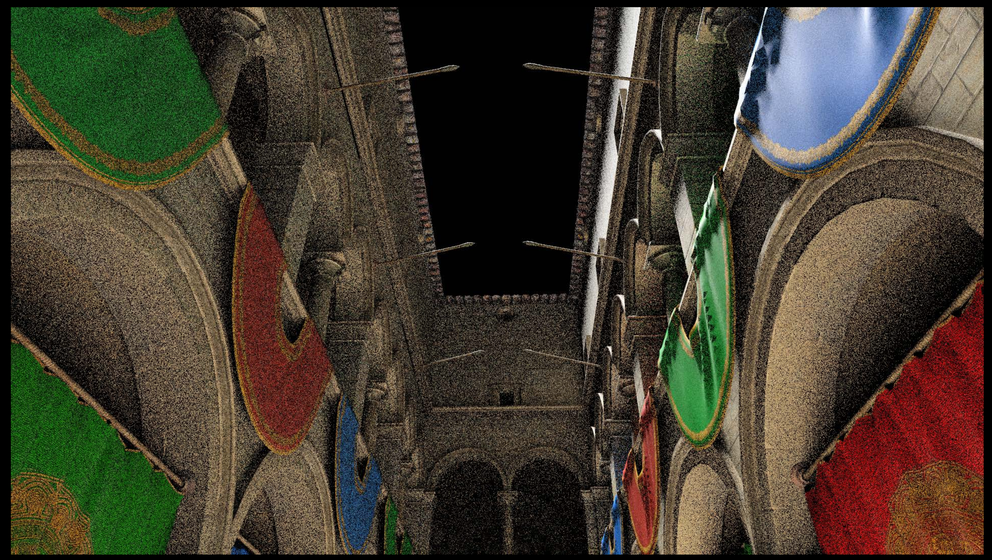
Create training graph

Run training

Deploy inference on new images

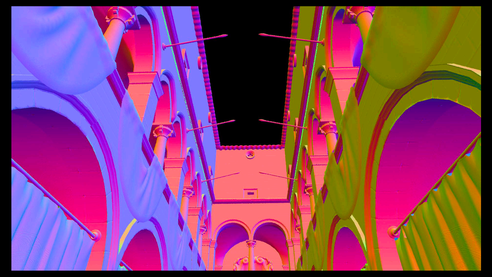
ML Denoising — Training Inputs

Example

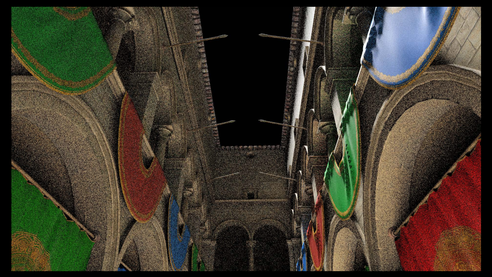


ML Denoising — Training Inputs

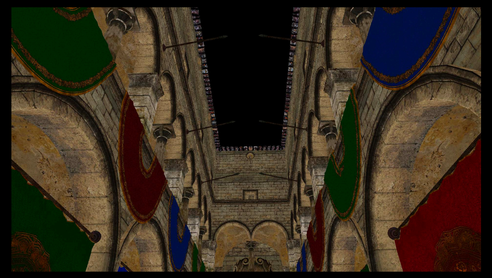
Example



Surface Normals



Noisy Image

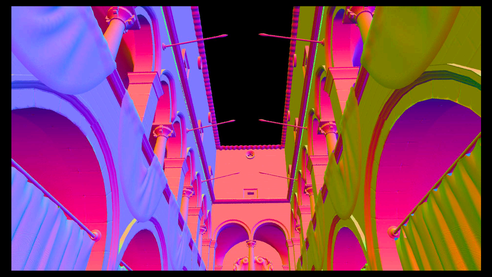


Albedo

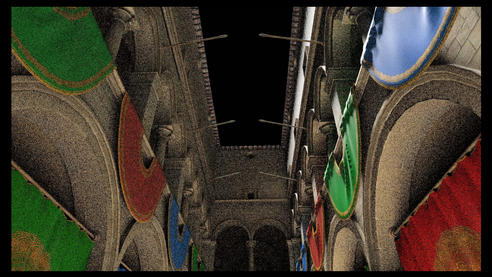
Clean Image

ML Denoising — Training Inputs

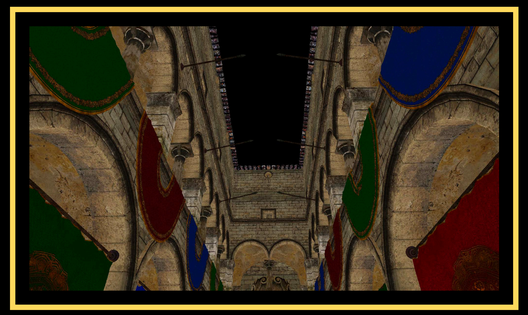
Example



Surface Normals



Noisy Image

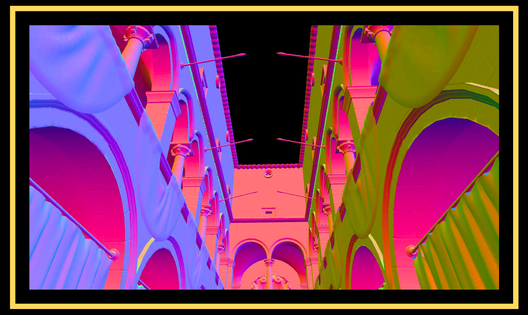


Albedo

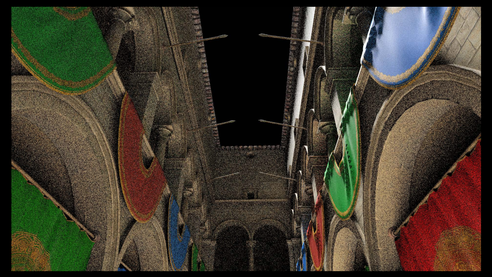
Clean Image

ML Denoising — Training Inputs

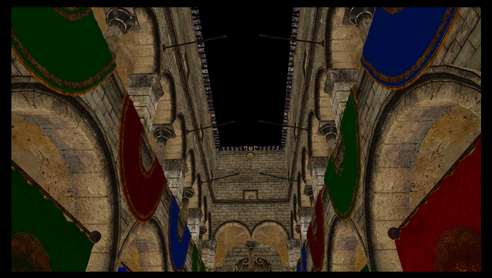
Example



Surface Normals



Noisy Image

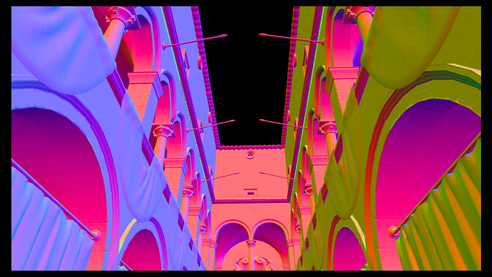


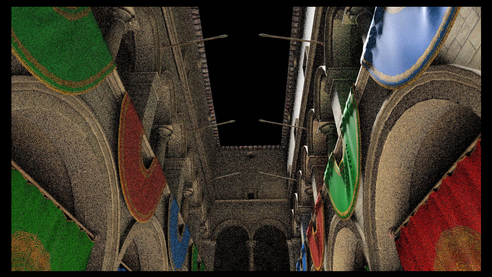
Albedo

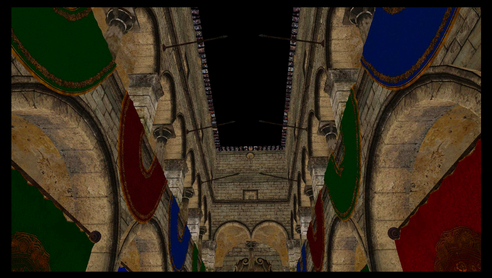
Clean Image

ML Denoising — Training Inputs

Example

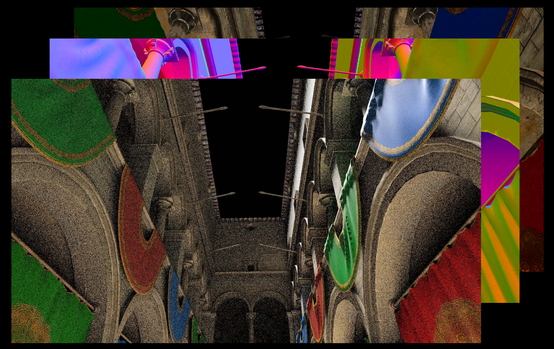






ML Denoising — Training Inputs

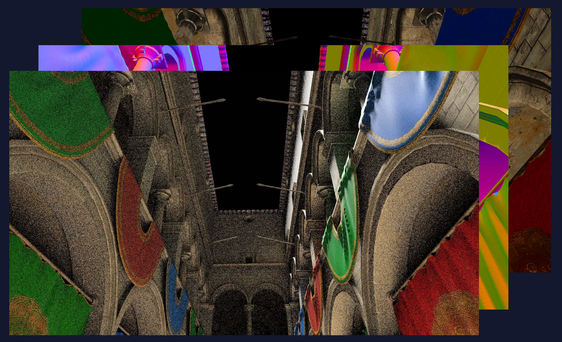
Example



ML Denoising — Training Inputs

Example

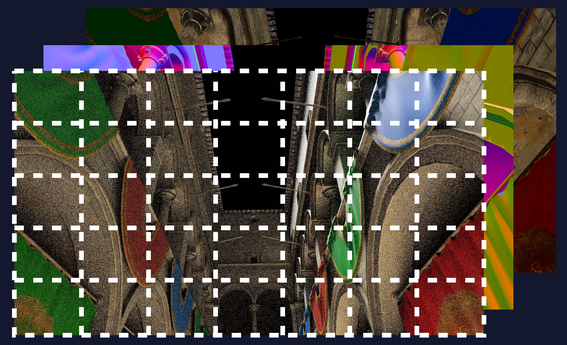
```
kernel void concat(texture2d_array<float, access::write> result      [[ texture(0) ]],
                   texture2d_array<float, access::read>  rgbImage    [[ texture(1) ]],
                   texture2d_array<float, access::read>  albedoImage [[ texture(2) ]],
                   texture2d_array<float, access::read>. normalImage [[ texture(3) ]],
                   ushort2   gid [[thread_position_in_grid]]){
   float4 image = rgbImage.read(gid);
   float4 albedo = albedoImage.read(gid);
   float4 normals = normalImage.read(gid);

   float4 pixel0 = float4(image.rgb, albedo.r);
   float4 pixel1 = float4(albedo.gb, normals.rg);
   float4 pixel2 = float4(normals.b, 0.0f, 0.0f, 0.0f);

   result.write(pixel0, gid, 0);
   result.write(pixel1, gid, 1);
   result.write(pixel2, gid, 2);}
```



ML Denoising — Training Inputs

Example

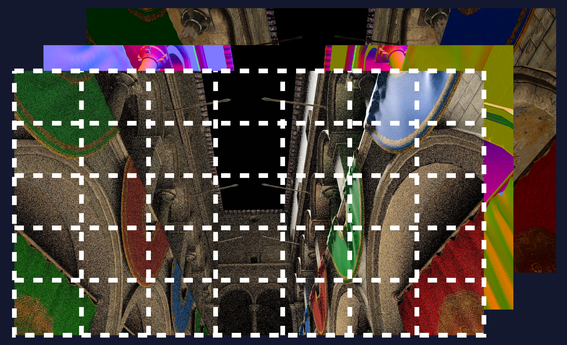
```
kernel void concat(texture2d_array<float, access::write> result      [[ texture(0) ]],
                   texture2d_array<float, access::read>  rgbImage    [[ texture(1) ]],
                   texture2d_array<float, access::read>  albedoImage [[ texture(2) ]],
                   texture2d_array<float, access::read>. normalImage [[ texture(3) ]],
                   ushort2   gid [[thread_position_in_grid]]){
   float4 image = rgbImage.read(gid);
   float4 albedo = albedoImage.read(gid);
   float4 normals = normalImage.read(gid);

   float4 pixel0 = float4(image.rgb, albedo.r);
   float4 pixel1 = float4(albedo.gb, normals.rg);
   float4 pixel2 = float4(normals.b, 0.0f, 0.0f, 0.0f);

   result.write(pixel0, gid, 0);
   result.write(pixel1, gid, 1);
   result.write(pixel2, gid, 2);}
```



ML Denoising — Training Inputs

Example

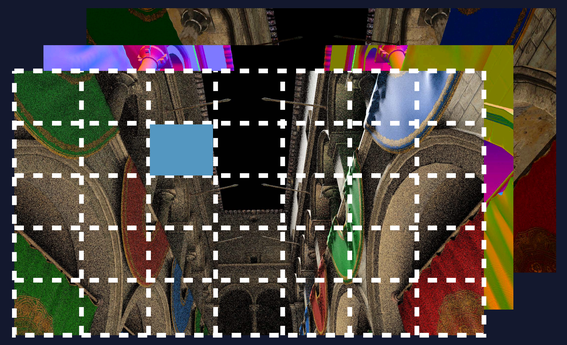
```
kernel void concat(texture2d_array<float, access::write> result      [[ texture(0) ]],
                   texture2d_array<float, access::read>  rgbImage    [[ texture(1) ]],
                   texture2d_array<float, access::read>  albedoImage [[ texture(2) ]],
                   texture2d_array<float, access::read>. normalImage [[ texture(3) ]],
                   ushort2   gid [[thread_position_in_grid]]){
   float4 image = rgbImage.read(gid);
   float4 albedo = albedoImage.read(gid);
   float4 normals = normalImage.read(gid);

   float4 pixel0 = float4(image.rgb, albedo.r);
   float4 pixel1 = float4(albedo.gb, normals.rg);
   float4 pixel2 = float4(normals.b, 0.0f, 0.0f, 0.0f);

   result.write(pixel0, gid, 0);
   result.write(pixel1, gid, 1);
   result.write(pixel2, gid, 2);}
```



ML Denoising — Training Inputs

Example

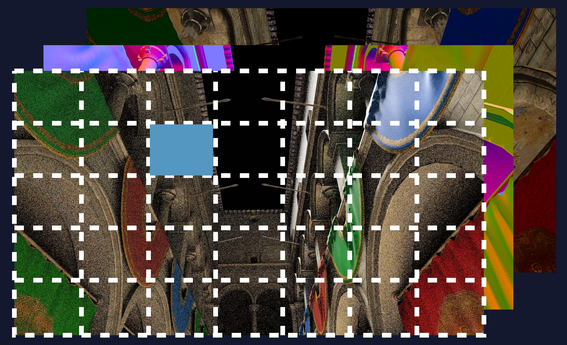
```
kernel void concat(texture2d_array<float, access::write> result      [[ texture(0) ]],
                   texture2d_array<float, access::read>  rgbImage    [[ texture(1) ]],
                   texture2d_array<float, access::read>  albedoImage [[ texture(2) ]],
                   texture2d_array<float, access::read>. normalImage [[ texture(3) ]],
                   ushort2   gid [[thread_position_in_grid]]){
    float4 image = rgbImage.read(gid);
    float4 albedo = albedoImage.read(gid);
    float4 normals = normalImage.read(gid);

    float4 pixel0 = float4(image.rgb, albedo.r);
    float4 pixel1 = float4(albedo.gb, normals.rg);
    float4 pixel2 = float4(normals.b, 0.0f, 0.0f, 0.0f);

    result.write(pixel0, gid, 0);
    result.write(pixel1, gid, 1);
    result.write(pixel2, gid, 2);}
```



ML Denoising — Training Inputs

Example

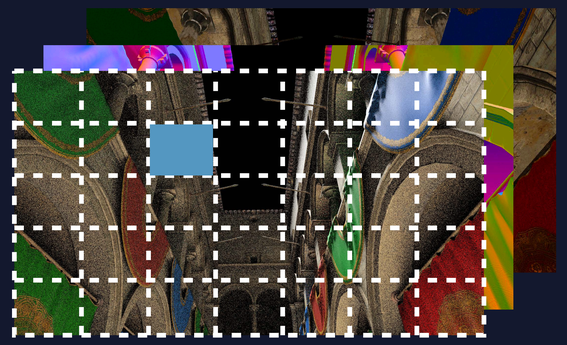
```
kernel void concat(texture2d_array<float, access::write> result      [[ texture(0) ]],
                   texture2d_array<float, access::read>  rgbImage    [[ texture(1) ]],
                   texture2d_array<float, access::read>  albedoImage [[ texture(2) ]],
                   texture2d_array<float, access::read>. normalImage [[ texture(3) ]],
                   ushort2   gid [[thread_position_in_grid]]){
    float4 image = rgbImage.read(gid);
    float4 albedo = albedoImage.read(gid);
    float4 normals = normalImage.read(gid);

    float4 pixel0 = float4(image.rgb, albedo.r);
    float4 pixel1 = float4(albedo.gb, normals.rg);
    float4 pixel2 = float4(normals.b, 0.0f, 0.0f, 0.0f);

    result.write(pixel0, gid, 0);
    result.write(pixel1, gid, 1);
    result.write(pixel2, gid, 2);}
```



ML Denoising — Training Inputs

Example

```
kernel void concat(texture2d_array<float, access::write> result      [[ texture(0) ]],
                   texture2d_array<float, access::read>  rgbImage    [[ texture(1) ]],
                   texture2d_array<float, access::read>  albedoImage [[ texture(2) ]],
                   texture2d_array<float, access::read>. normalImage [[ texture(3) ]],
                   ushort2   gid [[thread_position_in_grid]]){
    float4 image = rgbImage.read(gid);
    float4 albedo = albedoImage.read(gid);
    float4 normals = normalImage.read(gid);

    float4 pixel0 = float4(image.rgb, albedo.r);
    float4 pixel1 = float4(albedo.gb, normals.rg);
    float4 pixel2 = float4(normals.b, 0.0f, 0.0f, 0.0f);

    result.write(pixel0, gid, 0);
    result.write(pixel1, gid, 1);
    result.write(pixel2, gid, 2);}
```



```
// Encode concatenation kernel.
func encodeConcat(cmdBuf: MTLCommandBuffer,
                  rgbImage: MPSImage,
                  albedoImage: MPSImage,
                  normalsImage: MPSImage,
                  imageDims: MTLSize)-> MPSImage {
   let encoder = cmdBuf.makeComputeCommandEncoder()! // Create an encoder.
   …
   let result = MPSImage(device: cmdBuf.device, imageDescriptor: desc)
   // Set arguments for the kernel and dispatch.
   encoder.setTexture(result.texture, atIndex: 0)
   encoder.setTexture(rgbImage.texture, atIndex: 1)
   encoder.setTexture(albedoImage.texture, atIndex: 2)
   encoder.setTexture(normalsImage.texture, atIndex: 3)
   encoder.dispatchThreads(imageDims, threadsPerThreadgroup:MTLSize(width:16, height:16,
                                                                    depth:1))
   encoder.endEncoding()
   return result // Return the result image.
}
```

```
// Encode concatenation kernel.
func encodeConcat(cmdBuf: MTLCommandBuffer,
                  rgbImage: MPSImage,
                  albedoImage: MPSImage,
                  normalsImage: MPSImage,
                  imageDims: MTLSize)-> MPSImage {
    let encoder = cmdBuf.makeComputeCommandEncoder()! // Create an encoder.
    …
    let result = MPSImage(device: cmdBuf.device, imageDescriptor: desc)
    // Set arguments for the kernel and dispatch.
    encoder.setTexture(result.texture, atIndex: 0)
    encoder.setTexture(rgbImage.texture, atIndex: 1)
    encoder.setTexture(albedoImage.texture, atIndex: 2)
    encoder.setTexture(normalsImage.texture, atIndex: 3)
    encoder.dispatchThreads(imageDims, threadsPerThreadgroup:MTLSize(width:16, height:16,
                                                                     depth:1))
    encoder.endEncoding()
    return result // Return the result image.
}
```

```
// Encode concatenation kernel.
func encodeConcat(cmdBuf: MTLCommandBuffer,
                  rgbImage: MPSImage,
                  albedoImage: MPSImage,
                  normalsImage: MPSImage,
                  imageDims: MTLSize)-> MPSImage {
    let encoder = cmdBuf.makeComputeCommandEncoder()! // Create an encoder.
    …
    let result = MPSImage(device: cmdBuf.device, imageDescriptor: desc)
    // Set arguments for the kernel and dispatch.
    encoder.setTexture(result.texture, atIndex: 0)
    encoder.setTexture(rgbImage.texture, atIndex: 1)
    encoder.setTexture(albedoImage.texture, atIndex: 2)
    encoder.setTexture(normalsImage.texture, atIndex: 3)
    encoder.dispatchThreads(imageDims, threadsPerThreadgroup:MTLSize(width:16, height:16,
                                                                     depth:1))
    encoder.endEncoding()
    return result // Return the result image.
}
```

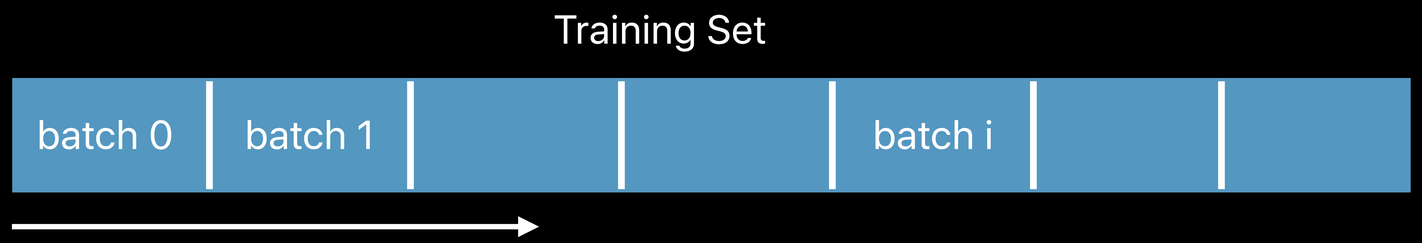
```
// Encode concatenation kernel.
func encodeConcat(cmdBuf: MTLCommandBuffer,
                  rgbImage: MPSImage,
                  albedoImage: MPSImage,
                  normalsImage: MPSImage,
                  imageDims: MTLSize)-> MPSImage {
    let encoder = cmdBuf.makeComputeCommandEncoder()! // Create an encoder.
    …
    let result = MPSImage(device: cmdBuf.device, imageDescriptor: desc)
    // Set arguments for the kernel and dispatch.
    encoder.setTexture(result.texture, atIndex: 0)
    encoder.setTexture(rgbImage.texture, atIndex: 1)
    encoder.setTexture(albedoImage.texture, atIndex: 2)
    encoder.setTexture(normalsImage.texture, atIndex: 3)
    encoder.dispatchThreads(imageDims, threadsPerThreadgroup:MTLSize(width:16, height:16,
                                                                     depth:1))
    encoder.endEncoding()
    return result // Return the result image.
}
```

```swift
// Encode concatenation kernel.
func encodeConcat(cmdBuf: MTLCommandBuffer,
                  rgbImage: MPSImage,
                  albedoImage: MPSImage,
                  normalsImage: MPSImage,
                  imageDims: MTLSize)-> MPSImage {
    let encoder = cmdBuf.makeComputeCommandEncoder()! // Create an encoder.
    …
    let result = MPSImage(device: cmdBuf.device, imageDescriptor: desc)
    // Set arguments for the kernel and dispatch.
    encoder.setTexture(result.texture, atIndex: 0)
    encoder.setTexture(rgbImage.texture, atIndex: 1)
    encoder.setTexture(albedoImage.texture, atIndex: 2)
    encoder.setTexture(normalsImage.texture, atIndex: 3)
    encoder.dispatchThreads(imageDims, threadsPerThreadgroup:MTLSize(width:16, height:16,
                                                                     depth:1))
    encoder.endEncoding()
    return result // Return the result image.
}
```

```
// Encode concatenation kernel.
func encodeConcat(cmdBuf: MTLCommandBuffer,
                  rgbImage: MPSImage,
                  albedoImage: MPSImage,
                  normalsImage: MPSImage,
                  imageDims: MTLSize)-> MPSImage {
    let encoder = cmdBuf.makeComputeCommandEncoder()! // Create an encoder.
    …
    let result = MPSImage(device: cmdBuf.device, imageDescriptor: desc)
    // Set arguments for the kernel and dispatch.
    encoder.setTexture(result.texture, atIndex: 0)
    encoder.setTexture(rgbImage.texture, atIndex: 1)
    encoder.setTexture(albedoImage.texture, atIndex: 2)
    encoder.setTexture(normalsImage.texture, atIndex: 3)
    encoder.dispatchThreads(imageDims, threadsPerThreadgroup:MTLSize(width:16, height:16,
                                                                     depth:1))
    encoder.endEncoding()
    return result // Return the result image.
}
```

ML Denoising — Training Loop

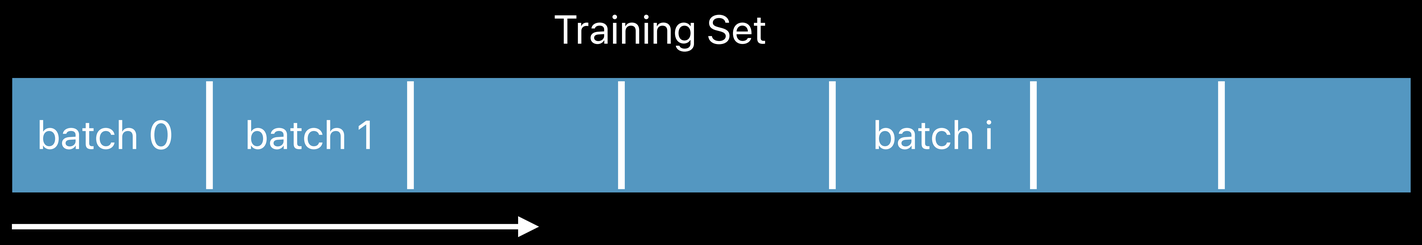
Example




```
func train(rgbImages: [MPSImage], albedoImages: [MPSImage]
        normalImages: [MPSImage], cleanImages: [MPSImage]) {
    let trainingImages = encodeConcat(rgbImages, albedoImages, normalImages) // For each image
    let inputs = [trainingImages, cleanImages]
    let cmdBuf = MPSCommandBuffer(from: queue)

    // Encode the graph.
    graph.encodeBatch(to: cmdBuf, sourceImages: inputs, sourceStates: nil];
    cmdBuf.commit()
}
```

ML Denoising — Training Loop

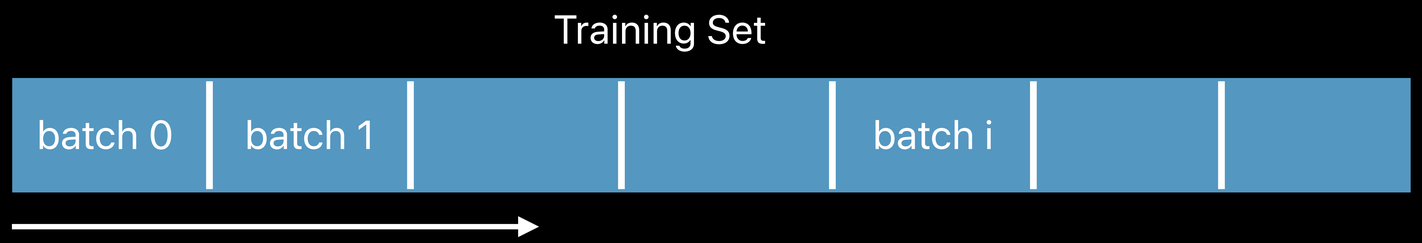
Example




```
func train(rgbImages: [MPSImage], albedoImages: [MPSImage]
        normalImages: [MPSImage], cleanImages: [MPSImage]) {
    let trainingImages = encodeConcat(rgbImages, albedoImages, normalImages) // For each image
    let inputs = [trainingImages, cleanImages]
    let cmdBuf = MPSCommandBuffer(from: queue)

    // Encode the graph.
    graph.encodeBatch(to: cmdBuf, sourceImages: inputs, sourceStates: nil];
    cmdBuf.commit()
}
```

ML Denoising — Training Loop

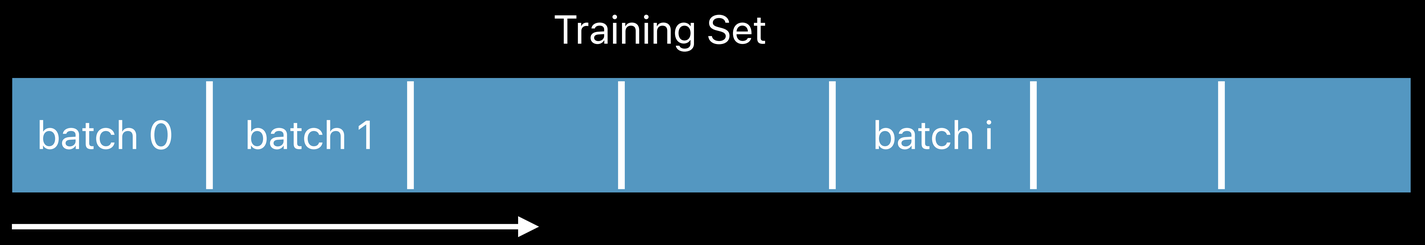
Example




```
func train(rgbImages: [MPSImage], albedoImages: [MPSImage]
        normalImages: [MPSImage], cleanImages: [MPSImage]) {
    let trainingImages = encodeConcat(rgbImages, albedoImages, normalImages) // For each image
    let inputs = [trainingImages, cleanImages]
    let cmdBuf = MPSCommandBuffer(from: queue)

    // Encode the graph.
    graph.encodeBatch(to: cmdBuf, sourceImages: inputs, sourceStates: nil];
    cmdBuf.commit()
}
```

ML Denoising — Training Loop

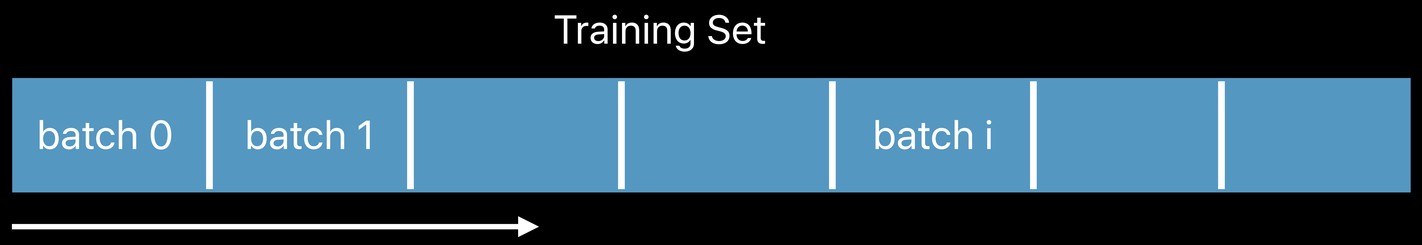
Example




```
func train(rgbImages: [MPSImage], albedoImages: [MPSImage]
        normalImages: [MPSImage], cleanImages: [MPSImage]) {
    let trainingImages = encodeConcat(rgbImages, albedoImages, normalImages) // For each image
    let inputs = [trainingImages, cleanImages]
    let cmdBuf = MPSCommandBuffer(from: queue)

    // Encode the graph.
    graph.encodeBatch(to: cmdBuf, sourceImages: inputs, sourceStates: nil];
    cmdBuf.commit()
}
```

ML Denoising — Training Loop

Example




```
func train(rgbImages: [MPSImage], albedoImages: [MPSImage]
        normalImages: [MPSImage], cleanImages: [MPSImage]) {
    let trainingImages = encodeConcat(rgbImages, albedoImages, normalImages) // For each image
    let inputs = [trainingImages, cleanImages]
    let cmdBuf = MPSCommandBuffer(from: queue)

    // Encode the graph.
    graph.encodeBatch(to: cmdBuf, sourceImages: inputs, sourceStates: nil];
    cmdBuf.commit()
}
```

ML Denoising — Training Loop

Example

Each epoch, process the training set

Every 100 epochs

- Update the training set
- Validate the network against the validation set

Every 1000 epochs decrease the learning rate

```
for epoch in 0..<NUM_EPOCHS {
   for batch in 0..<NUM_IMAGE_BATCHES {
      denoiserGraph.train(images:trainingImages.rgb[batch],
                          albedo:trainingImages.albedo[batch],
                         normals:trainingImages.normals[batch], cleanImages: cleanImages)}

   if (epoch % 100 == 0){
      trainingImages = load(trainingData[epoch])
      cleanImages = load(targetData[epoch])

      float validationLoss = 0.0
      for image in 0..<NUM_VALIDATION_IMAGES {
         let result = denoiserGraph.inference(images: validationImages[image])
         validationLoss += loss(result, validationTargets[image])
      }
   }
   if (epoch % 1000 == 0) {learningRate *= 0.5}
}
```

```
for epoch in 0..<NUM_EPOCHS {
   for batch in 0..<NUM_IMAGE_BATCHES {
      denoiserGraph.train(images:trainingImages.rgb[batch],
                          albedo:trainingImages.albedo[batch],
                         normals:trainingImages.normals[batch], cleanImages: cleanImages)}

   if (epoch % 100 == 0){
      trainingImages = load(trainingData[epoch])
      cleanImages = load(targetData[epoch])

      float validationLoss = 0.0
      for image in 0..<NUM_VALIDATION_IMAGES {
         let result = denoiserGraph.inference(images: validationImages[image])
         validationLoss += loss(result, validationTargets[image])
      }
   }
   if (epoch % 1000 == 0) {learningRate *= 0.5}
}
```

```
for epoch in 0..<NUM_EPOCHS {
   for batch in 0..<NUM_IMAGE_BATCHES {
      denoiserGraph.train(images:trainingImages.rgb[batch],
                          albedo:trainingImages.albedo[batch],
                         normals:trainingImages.normals[batch], cleanImages: cleanImages)}

   if (epoch % 100 == 0){
      trainingImages = load(trainingData[epoch])
      cleanImages = load(targetData[epoch])

      float validationLoss = 0.0
      for image in 0..<NUM_VALIDATION_IMAGES {
         let result = denoiserGraph.inference(images: validationImages[image])
         validationLoss += loss(result, validationTargets[image])
      }
   }
   if (epoch % 1000 == 0) {learningRate *= 0.5}
}
```

```
for epoch in 0..<NUM_EPOCHS {
   for batch in 0..<NUM_IMAGE_BATCHES {
      denoiserGraph.train(images:trainingImages.rgb[batch],
                          albedo:trainingImages.albedo[batch],
                         normals:trainingImages.normals[batch], cleanImages: cleanImages)}

   if (epoch % 100 == 0){
      trainingImages = load(trainingData[epoch])
      cleanImages = load(targetData[epoch])

      float validationLoss = 0.0
      for image in 0..<NUM_VALIDATION_IMAGES {
         let result = denoiserGraph.inference(images: validationImages[image])
         validationLoss += loss(result, validationTargets[image])
      }
   }
   if (epoch % 1000 == 0) {learningRate *= 0.5}
}
```

ML Denoising

Example

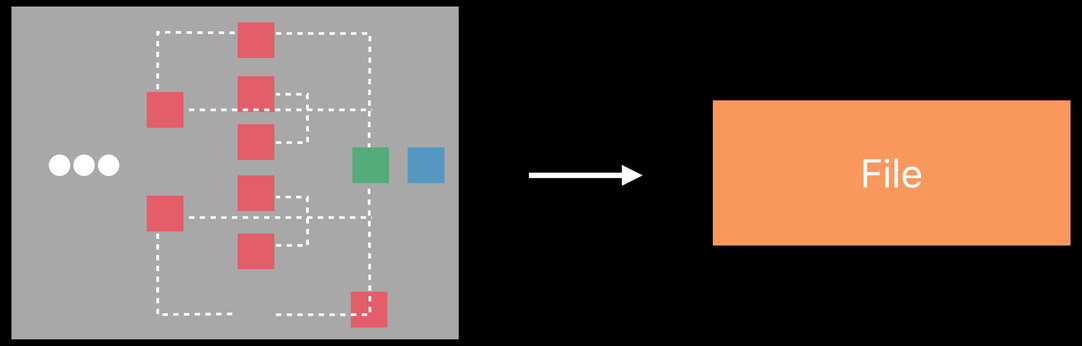
Create training graph

Run training

Deploy inference on new images

MPSNNGraph and Secure Coding

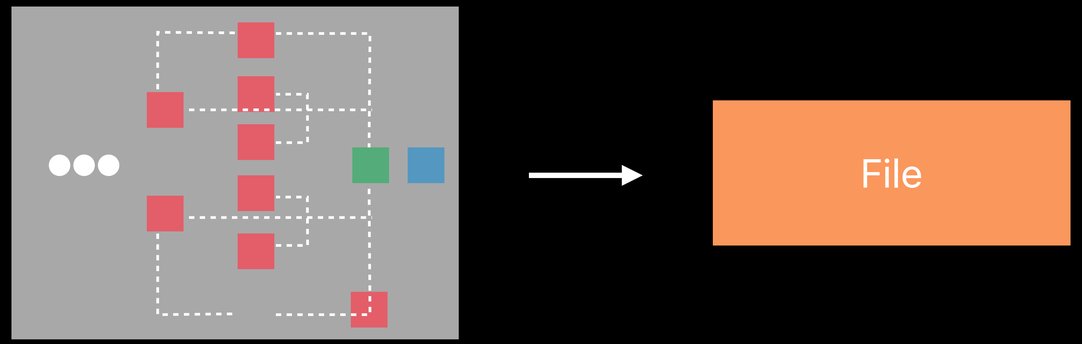
Implement secure coding on the data sources




```
class denoiserWeights: NSObject, MPSCNNConvolutionDataSource {
…
func supportsSecureCoding -> Bool { return true; }
func init(coder: NSCoder) -> Self {…}
func encode(coder: NSCoder) {…}
…
}
```

MPSNNGraph and Secure Coding

Implement secure coding on the data sources

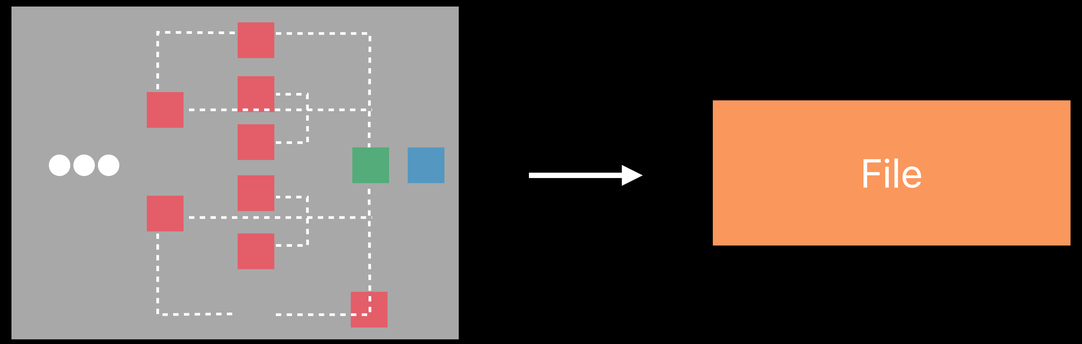



```
class denoiserWeights: NSObject, MPSCNNConvolutionDataSource {
…
func supportsSecureCoding -> Bool { return true; }
func init(coder: NSCoder) -> Self {…}
func encode(coder: NSCoder) {…}
…
}
```

ML Denoising

Example

Encode and save the graph

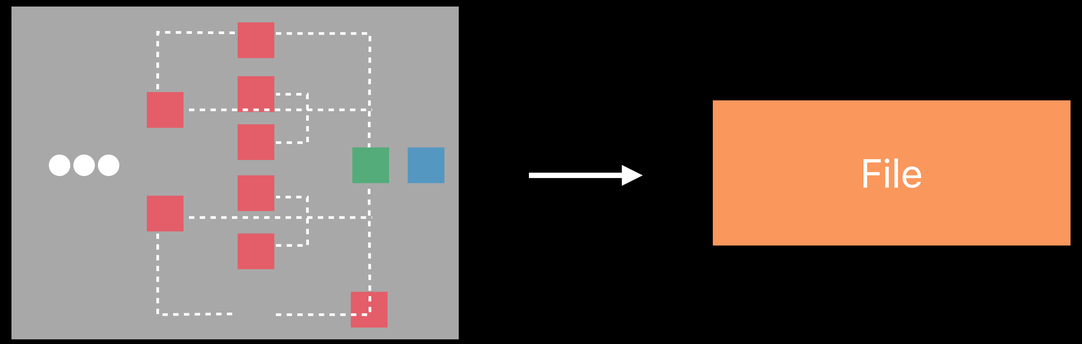



```
let coder = NSKeyedArchiver(requiringSecureCoding: true)

inferenceGraph.encode(with: coder)

let data = coder.encodedData
try! data.write(to: URL(string: “serializedGraph.mps“)!)
```

ML Denoising

Example

Encode and save the graph

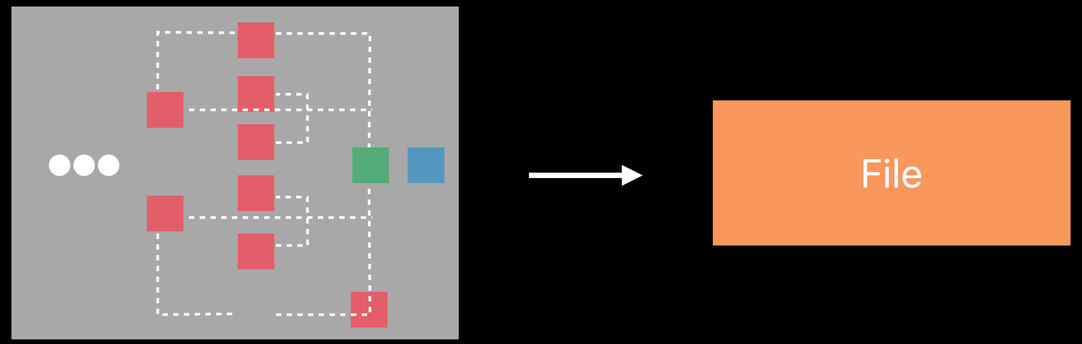



```
let coder = NSKeyedArchiver(requiringSecureCoding: true)

inferenceGraph.encode(with: coder)

let data = coder.encodedData
try! data.write(to: URL(string: “serializedGraph.mps“)!)
```

ML Denoising

Example

Encode and save the graph

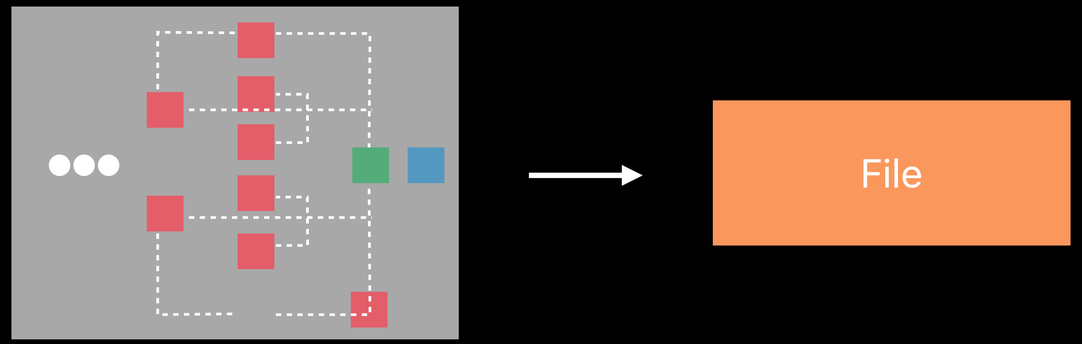



```
let coder = NSKeyedArchiver(requiringSecureCoding: true)

inferenceGraph.encode(with: coder)

let data = coder.encodedData
try! data.write(to: URL(string: “serializedGraph.mps“)!)
```

ML Denoising

Example

Encode and save the graph




```
let coder = NSKeyedArchiver(requiringSecureCoding: true)

inferenceGraph.encode(with: coder)

let data = coder.encodedData
try! data.write(to: URL(string: “serializedGraph.mps“)!)
```

ML Denoising

Example

MPSKeyedUnarchiver

Restore and execute the graph

```
let data = try Data(contentsOf: URL(string: “serializedGraph.mps”)!)

let coder = MPSKeyedUnarchiver(forReadingFrom: data, device: device, error: nil)
let inferenceGraph = MPSNNGraph(coder: coder)

let denoisedImage = inferenceGraph.encode(to: cmdBuf,
                                          sourceImages: [noisyImage])
```

ML Denoising

Example

MPSKeyedUnarchiver

Restore and execute the graph

```
let data = try Data(contentsOf: URL(string: “serializedGraph.mps”)!)

let coder = MPSKeyedUnarchiver(forReadingFrom: data, device: device, error: nil)
let inferenceGraph = MPSNNGraph(coder: coder)

let denoisedImage = inferenceGraph.encode(to: cmdBuf,
                                          sourceImages: [noisyImage])
```

ML Denoising

Example

MPSKeyedUnarchiver

Restore and execute the graph

```
let data = try Data(contentsOf: URL(string: “serializedGraph.mps”)!)

let coder = MPSKeyedUnarchiver(forReadingFrom: data, device: device, error: nil)
let inferenceGraph = MPSNNGraph(coder: coder)

let denoisedImage = inferenceGraph.encode(to: cmdBuf,
                                          sourceImages: [noisyImage])
```

ML Denoising

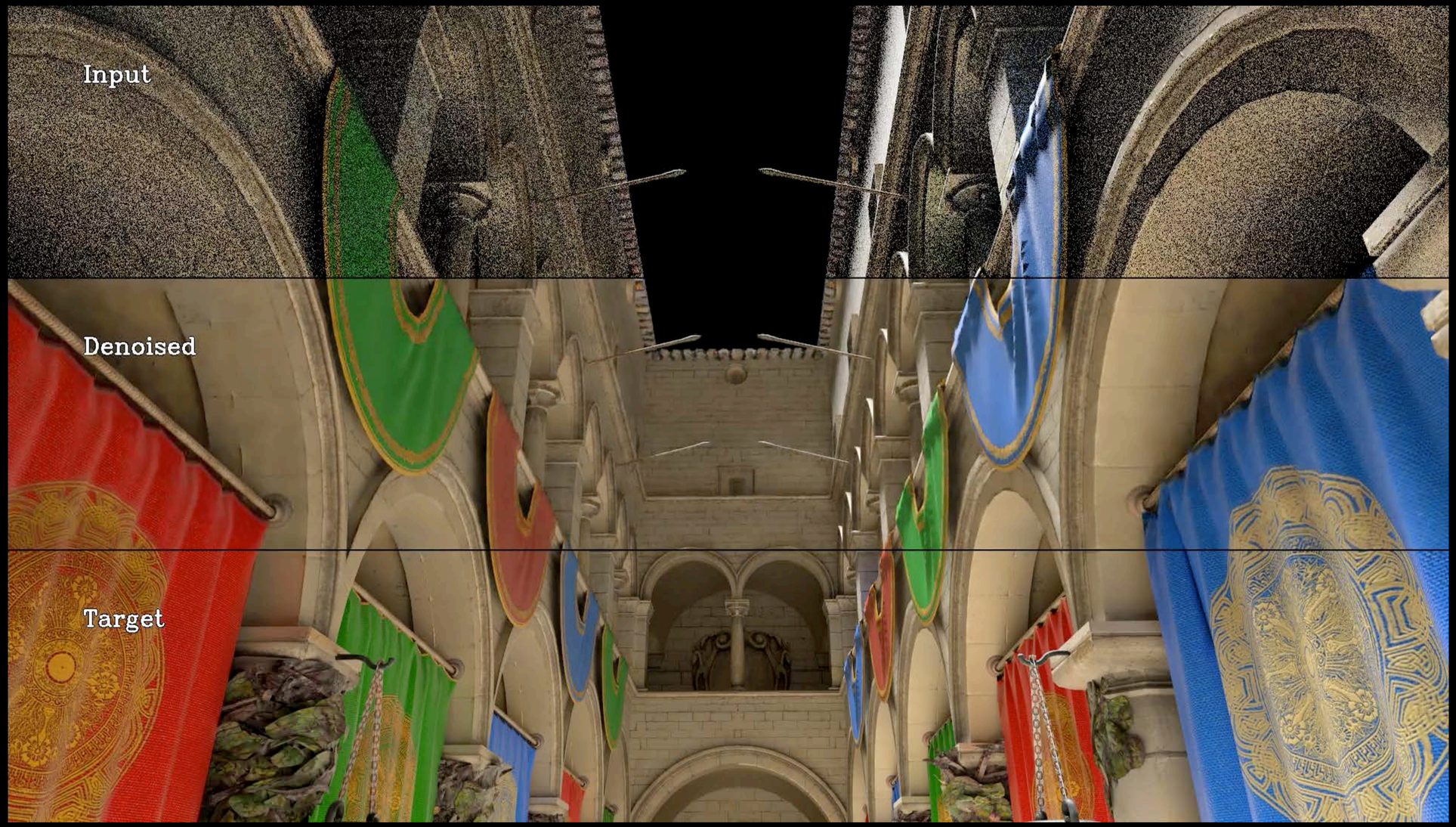
Example

MPSKeyedUnarchiver

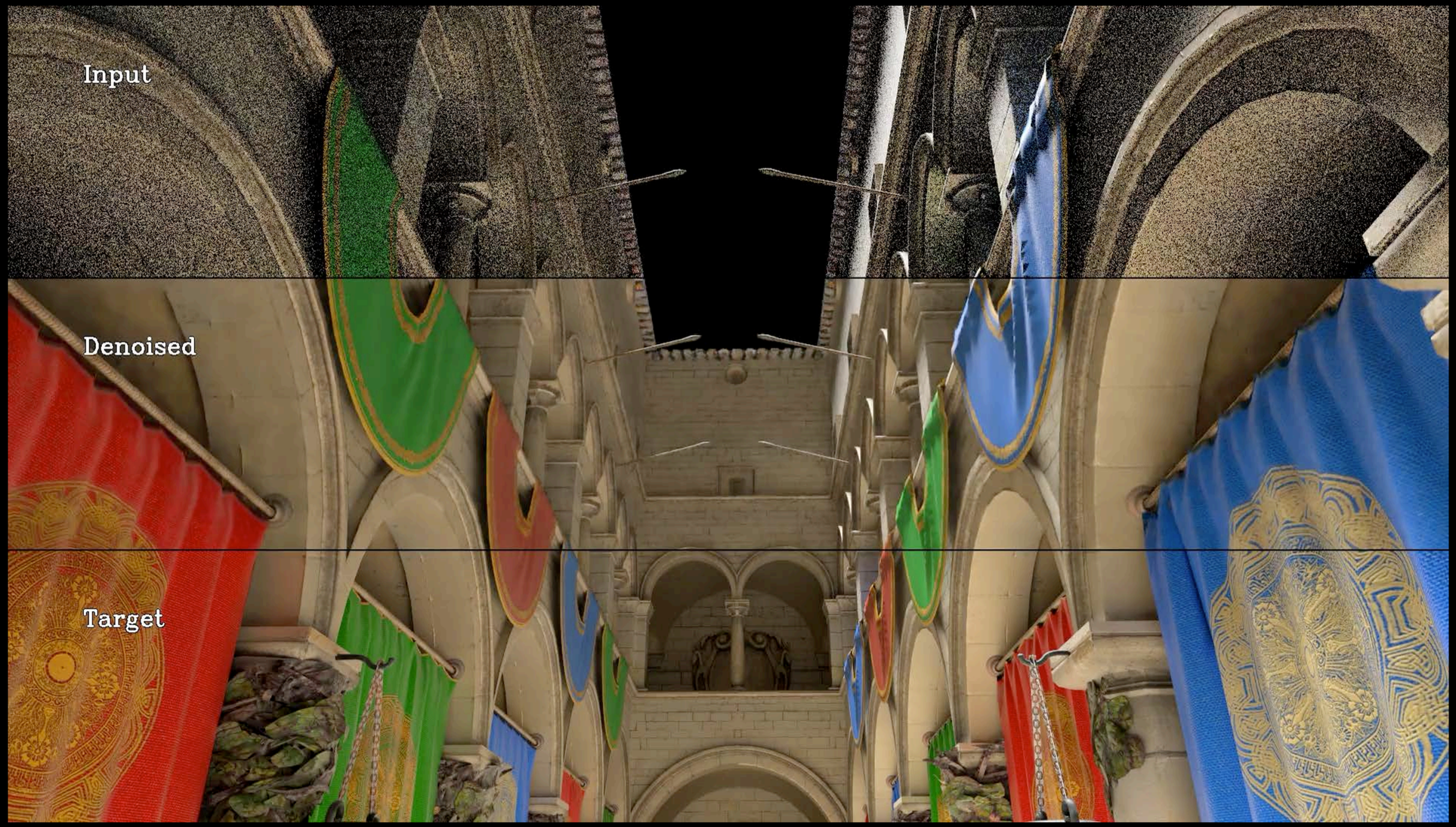
Restore and execute the graph

```
let data = try Data(contentsOf: URL(string: “serializedGraph.mps”)!)

let coder = MPSKeyedUnarchiver(forReadingFrom: data, device: device, error: nil)
let inferenceGraph = MPSNNGraph(coder: coder)

let denoisedImage = inferenceGraph.encode(to: cmdBuf,
                                          sourceImages: [noisyImage])
```

ML Denoising in Action



ML Denoising in Action



Summary

Supports training and inference on more networks

Improved performance and efficiency

Implicit graph creation

More Information

developer.apple.com/wwdc19/614


WWDC19