Cooperative Vectors in Gigi

Cooperative vectors are a preview feature of directX which simplify and accelerate large matrix multiplies and similar operations. Their goal is to aid machine learning workloads, but as machine learning is just a big pile of linear algebra, there are many other uses too, including convolution (image filtering!).

A high level overview of cooperative vectors is at: https://devblogs.microsoft.com/directx/enabling-neural-rendering-in-directx-cooperative-vector-support-coming-soon/

As this is a preview feature of DX12, the final feature may have a different interface. That means that Gigi too may have a different interface. Any techniques made with the cooperative vector interface as it is now may not work in the future without modification. Cooperative vectors are a viewer only feature at the moment, and not supported by any code generation targets yet.

Intel has a nice demo using cooperative vectors to do neural texture synthesis. https://github.com/GameTechDev/TextureSetNeuralCompressionSample

Deeper technical information about cooperative vectors in DX12 is at: https://devblogs.microsoft.com/directx/cooperative-vector/

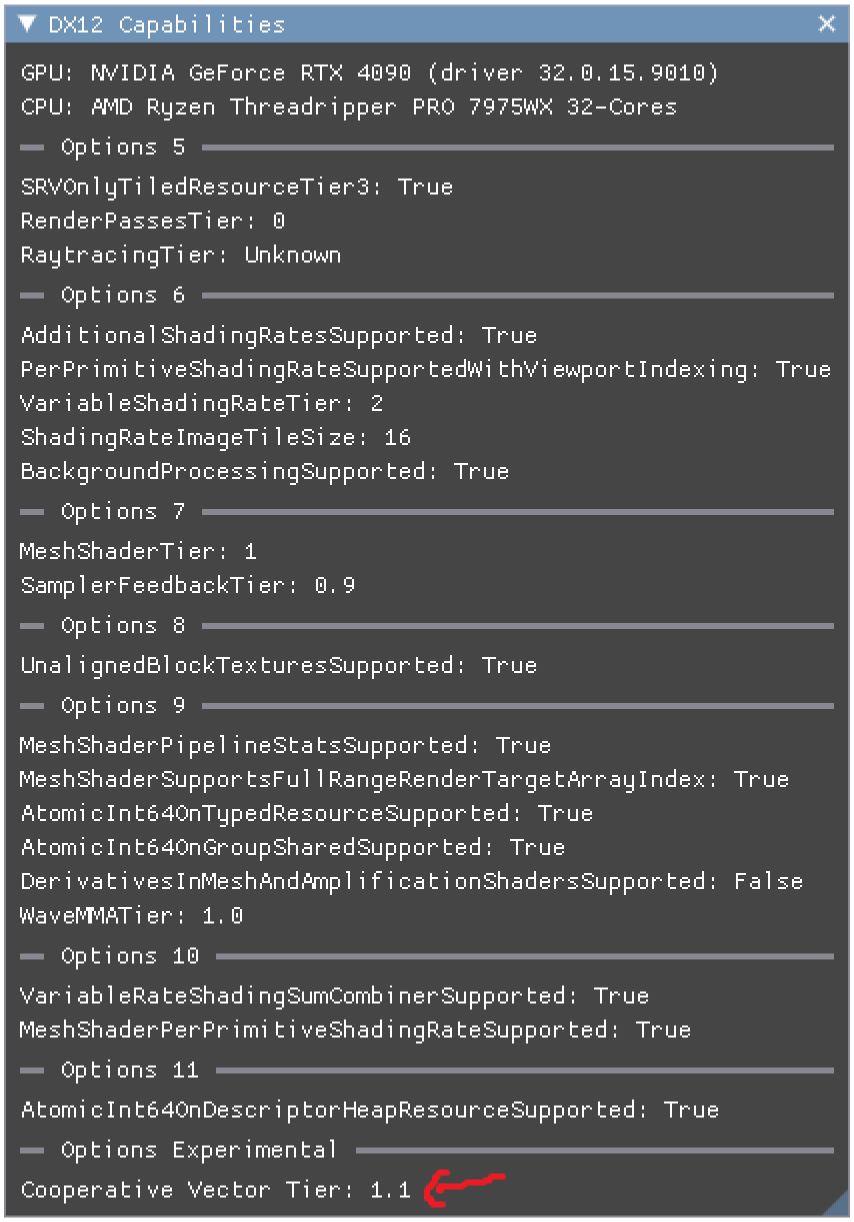
Apparently, the cooperative does (or can?) use tensor cores on the GPU when available.

Setup

To get cooperative vectors working in Gigi you must...

- Get special drivers that support them. (https://devblogs.microsoft.com/directx/cooperative-vector/#get-running)
- Turn on developer mode (https://learn.microsoft.com/en-us/windows/apps/get-started/enable-your-device-for-development)
- Run the Gigi Viewer with the command line parameters -AgilitySDKPreview -nopixcapture -norenderdoc.
    - **-AgilitySDKPreview** – This tells the viewer to use the preview version of directx which supports cooperative vectors.
    - **-nopixcapture -norenderdoc** – This turns off pix and renderdoc captures. These interfere with cooperative vector support, likely because they hook the DX12 API and so break some of this future functionality.

You can see if these steps were successful in the viewer by going to the View menu and selecting DX12 Capabilities.   At the bottom of the window you will see the Cooperative Vector Tier. If it says “None” or “Unknown”, you either did the steps wrong, or your computer does not support cooperative vectors.

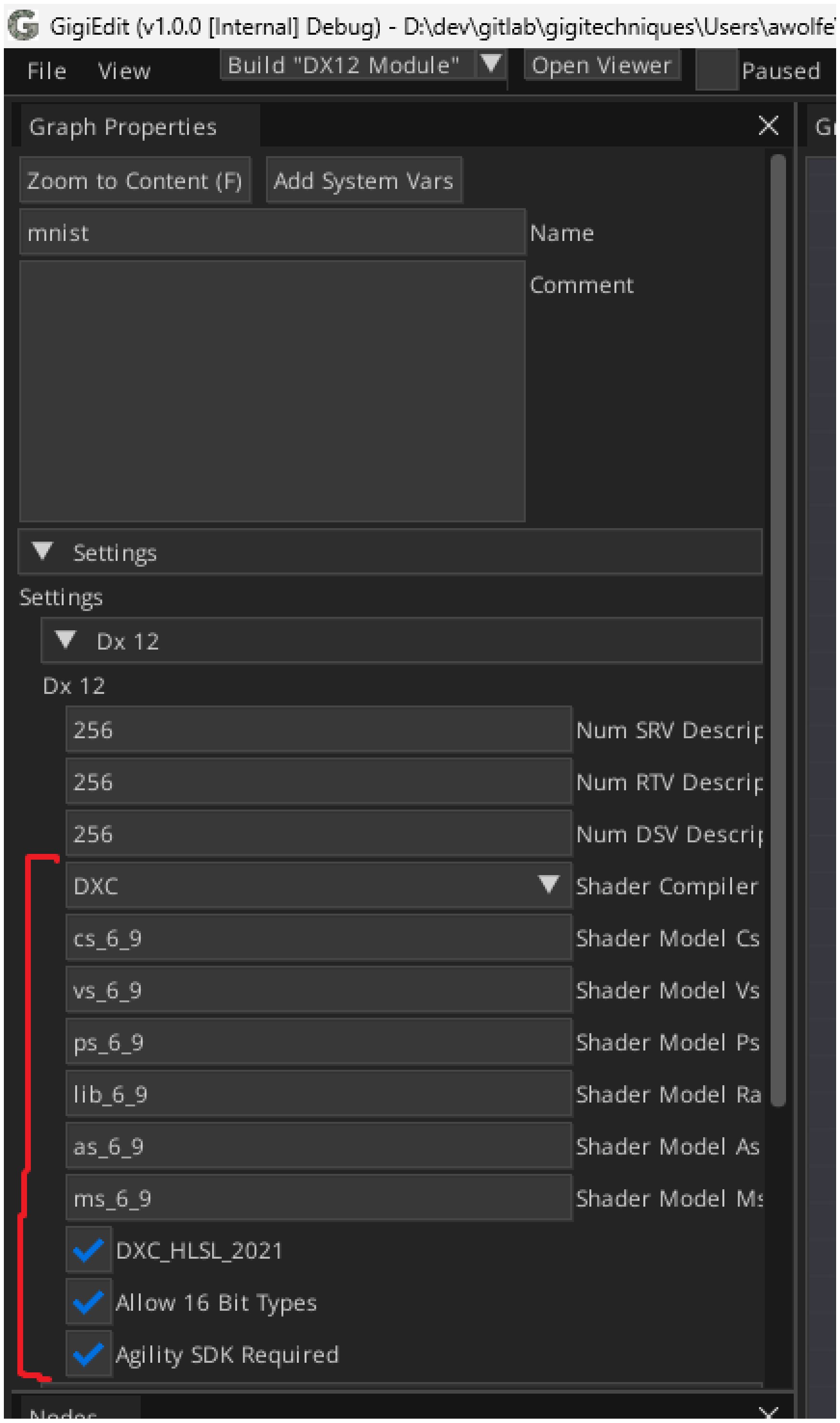


```
GPU: NVIDIA GeForce RTX 4090 (driver 32.0.15.9010)
CPU: AMD Ryzen Threadripper PRO 7975WX 32-Cores
— Options 5 ————————————————————————
SRVOnlyTiledResourceTier3: True
RenderPassesTier: 0
RaytracingTier: Unknown
— Options 6 ————————————————————————
AdditionalShadingRatesSupported: True
PerPrimitiveShadingRateSupportedWithViewportIndexing: True
VariableShadingRateTier: 2
ShadingRateImageTileSize: 16
BackgroundProcessingSupported: True
— Options 7 ————————————————————————
MeshShaderTier: 1
SamplerFeedbackTier: 0.9
— Options 8 ————————————————————————
UnalignedBlockTexturesSupported: True
— Options 9 ————————————————————————
MeshShaderPipelineStatsSupported: True
MeshShaderSupportsFullRangeRenderTargetArrayIndex: True
AtomicInt64OnTypedResourceSupported: True
AtomicInt64OnGroupSharedSupported: True
DerivativesInMeshAndAmplificationShadersSupported: False
WaveMMATier: 1.0
— Options 10 ———————————————————————
VariableRateShadingSumCombinerSupported: True
MeshShaderPerPrimitiveShadingRateSupported: True
— Options 11 ———————————————————————
AtomicInt64OnDescriptorHeapResourceSupported: True
— Options Experimental —————————————
Cooperative Vector Tier: 1.1
```

Cooperative vector tier 1.0 supports MatrixVectorMul and MatrixVectorMulAdd shader intrinsics. Tier 1.1 supports OuterProductAccumulate and VectorAccumulate.

Setting Up A Gigi Project

Cooperative vectors require using the DXC compiler, using shader model 6.9, the Agility SDK, and you are probably going to want 16 bit types, as well as HLSL 2021. You can set these things up in the "Graph Properties" window under **Settings->DX12**.


GigiEdit (v1.0.0 [Internal] Debug) - D:\dev\gitlab\gigitechniques\Users\awolfe

mnist Name

Comment

Settings

Dx 12

256 Num SRV Descrip

256 Num RTV Descrip

256 Num DSV Descrip

DXC Shader Compiler

cs_6_9 Shader Model Cs

vs_6_9 Shader Model Vs

ps_6_9 Shader Model Ps

lib_6_9 Shader Model Ra

as_6_9 Shader Model As

ms_6_9 Shader Model Ms

✓ DXC_HLSL_2021

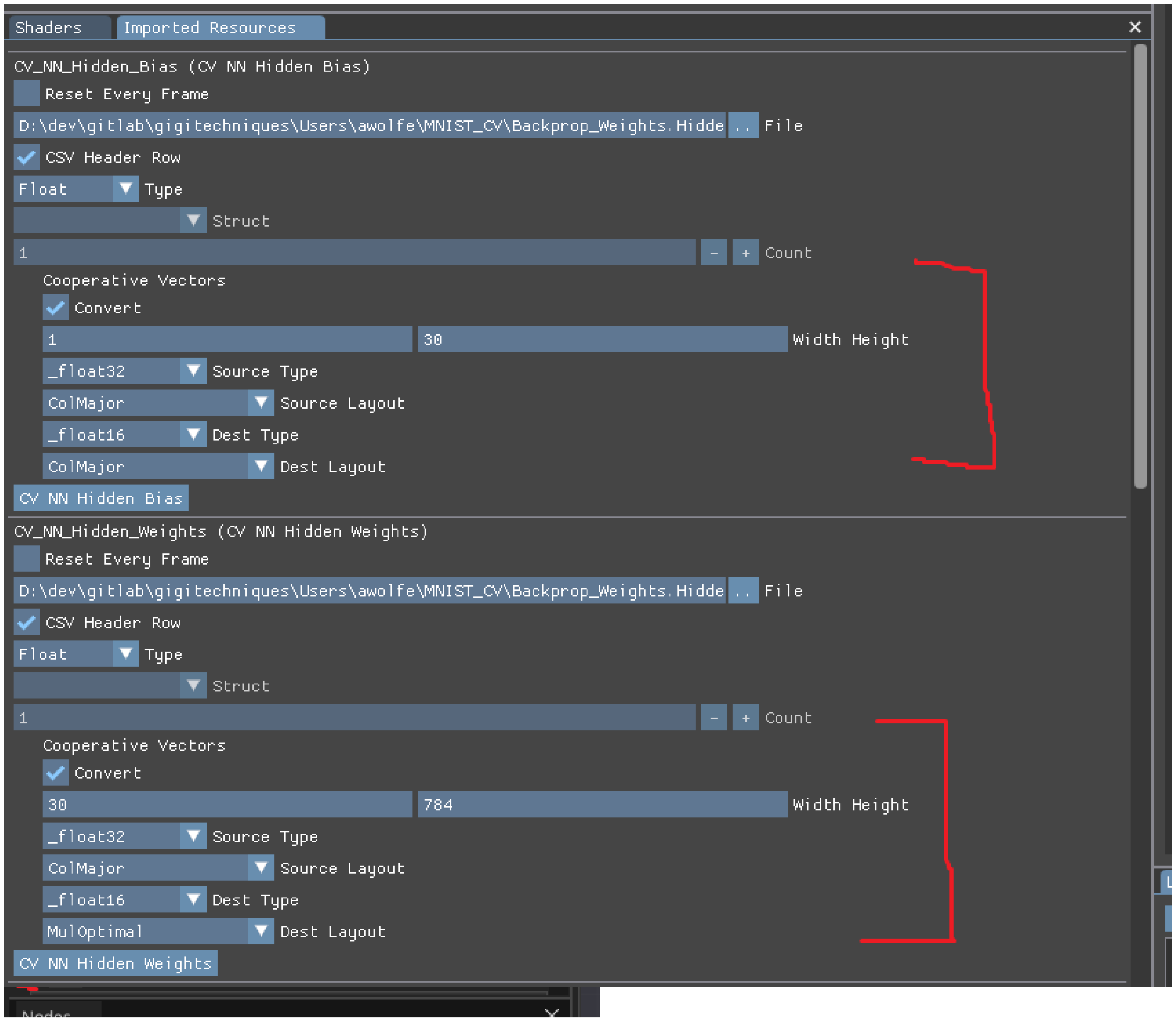
✓ Allow 16 Bit Types

✓ Agility SDK Required

Setting Up Data Buffers

Cooperative vectors support a variety of data formats including 16 bit floats (halfs), uint8s, and some exotic 8 bit floating point formats. When doing operations with this data, you also have to specify what layout the data is in: row major, column major, optimized for multiplication or optimized for outer products.

To help you get the data into these formats and layouts, a DX12 API is exposed, which Gigi exposes to you in the viewer for imported resources.

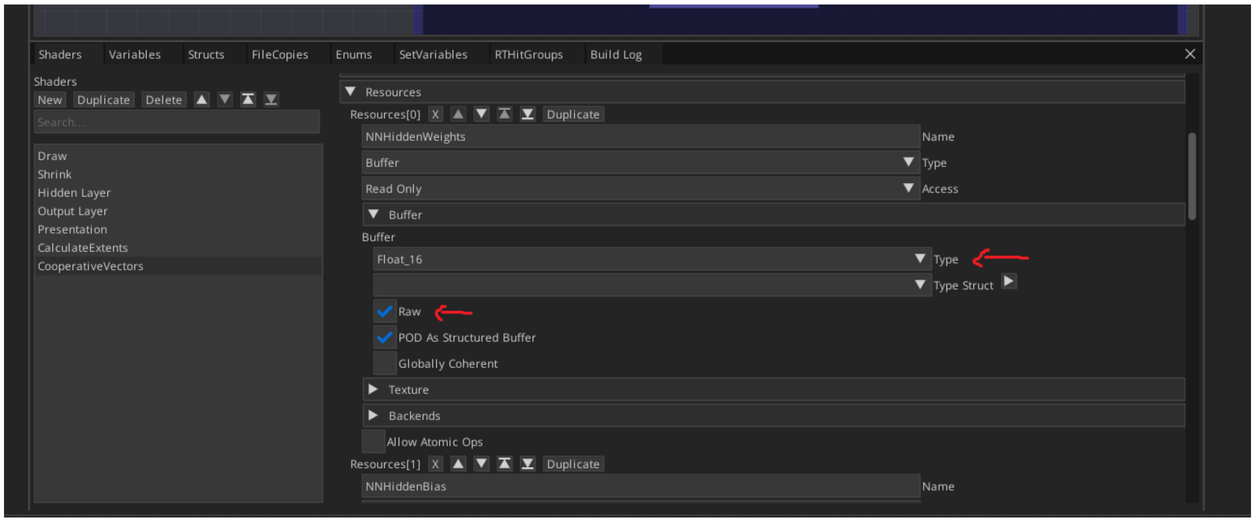


```
CV_NN_Hidden_Bias (CV NN Hidden Bias)
[ ] Reset Every Frame
D:\dev\gitlab\gigitechniques\Users\awolfe\MNIST_CV\Backprop_Weights.Hidde .. File
[✓] CSV Header Row
Float      Type
           Struct
1          - + Count
  Cooperative Vectors
  [✓] Convert
  1        30        Width Height
  _float32   Source Type
  ColMajor   Source Layout
  _float16   Dest Type
  ColMajor   Dest Layout
CV NN Hidden Bias

CV_NN_Hidden_Weights (CV NN Hidden Weights)
[ ] Reset Every Frame
D:\dev\gitlab\gigitechniques\Users\awolfe\MNIST_CV\Backprop_Weights.Hidde .. File
[✓] CSV Header Row
Float      Type
           Struct
1          - + Count
  Cooperative Vectors
  [✓] Convert
  30       784       Width Height
  _float32   Source Type
  ColMajor   Source Layout
  _float16   Dest Type
  MulOptimal Dest Layout
CV NN Hidden Weights
```

I hit some crashes when trying different layouts and formats, but following the path that the intel demo followed (16 bit floats, and "MulOptimal" layout for matrices) that things were a lot more stable. One crash I hit was that the 32 bit float format isn't supported. Your mileage may vary with different driver vendors or future driver versions.

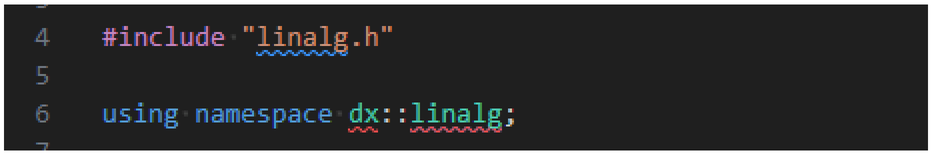
Writing A Shader

Using cooperative vectors in a shader is pretty easy luckily. The buffer resources involved need to be passed as **Raw** (ByteAddressBuffer or RWByteAddressBuffer), and you should still specify a type in the **Type** drop down.



Resources[0]
Name: NNHiddenWeights
Type: Buffer
Access: Read Only
Buffer
Type: Float_16
Type Struct:
[✓] Raw
[✓] POD As Structured Buffer
[ ] Globally Coherent
[ ] Allow Atomic Ops
Resources[1]
Name: NNHiddenBias

In the shader itself you need to include a shader header "linalg.h". Gigi handles putting that file into your shader folder. I prefer a "using namespace" line to make less typing.



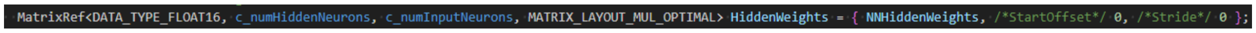
```
#include "linalg.h"

using namespace dx::linalg;
```

The cooperative vector shader API gives 4 functions

- Mul(MatrixRef M, InterpretedVector V) – Result = M*V
  - Just a matrix / vector multiply.
- MulAdd(MatrixRef M, InterpretedVector V1, VectorRef V2) – Result = M*V1+V2
  - This is useful for running a neural network layer: Weights*Input+Bias.
- OuterProductAccumulate(vector V1, vector V2, RWMatrixRef M)
  - M += V1 ⊗ V2
- VectorAccumulate(vector V, RWByteAddressBuffer Buffer, uint Offset)
  - Vector(Buffer[Offset]) += V

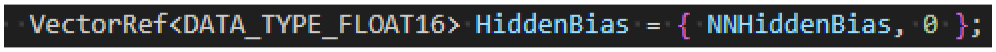
For the data types it exposes, MatrixRef and VectorRef (and RW variation which allows writing) don't actually load the data into memory but are just wrappers to let the functions load the data as needed. The InterpretedVector is a thin wrapper to a vector (called "long vectors" in DX12 terminology) and vectors are fully loaded into memory.

To get a MatrixRef, you tell it the data type, the rows and columns, and the layout as template parameter. You also have to give it a raw buffer (ByteAddressBuffer), the location in the buffer where the matrix begins, and a stride.



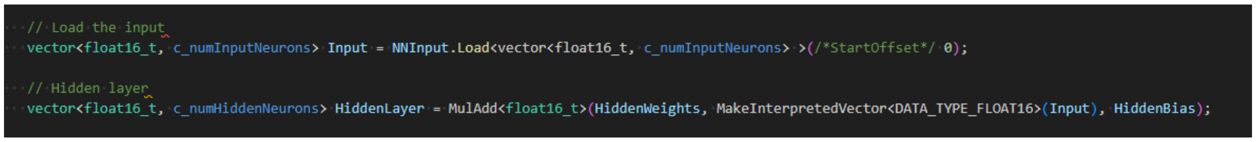
```
MatrixRef<DATA_TYPE_FLOAT16, c_numHiddenNeurons, c_numInputNeurons, MATRIX_LAYOUT_MUL_OPTIMAL> HiddenWeights = { NNHiddenWeights, /*StartOffset*/ 0, /*Stride*/ 0 };
```

To get a VectorRef, you only need to give a data type as the template parameter, and you give a raw Buffer and the location in the buffer where the vector begins.



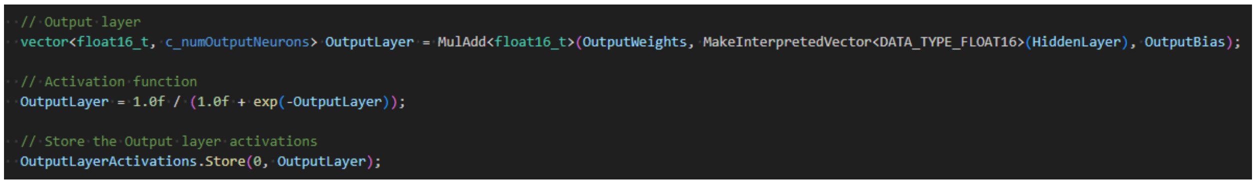
```
VectorRef<DATA_TYPE_FLOAT16> HiddenBias = { NNHiddenBias, 0 };
```

To get an InterpretedVector, you have to load data into a vector and then call MakeInterpretedVector(). In this snippet, NNInput is a raw buffer too.



```
    // Load the input
    vector<float16_t, c_numInputNeurons> Input = NNInput.Load<vector<float16_t, c_numInputNeurons> >(/*StartOffset*/ 0);

    // Hidden layer
    vector<float16_t, c_numHiddenNeurons> HiddenLayer = MulAdd<float16_t>(HiddenWeights, MakeInterpretedVector<DATA_TYPE_FLOAT16>(Input), HiddenBias);
```

The functions Mul and MulAdd return a vector, as shown in the snippet above. If you want to write these out, you can do so with 1 store statement to a raw buffer accessed as read/write (UAV access to a RWByteAddressBuffer). Doing activation functions to vectors is also super simple and written like regular shader code, which is very nice.



```
  // Output layer
  vector<float16_t, c_numOutputNeurons> OutputLayer = MulAdd<float16_t>(OutputWeights, MakeInterpretedVector<DATA_TYPE_FLOAT16>(HiddenLayer), OutputBias);

  // Activation function
  OutputLayer = 1.0f / (1.0f + exp(-OutputLayer));

  // Store the Output layer activations
  OutputLayerActivations.Store(0, OutputLayer);
```

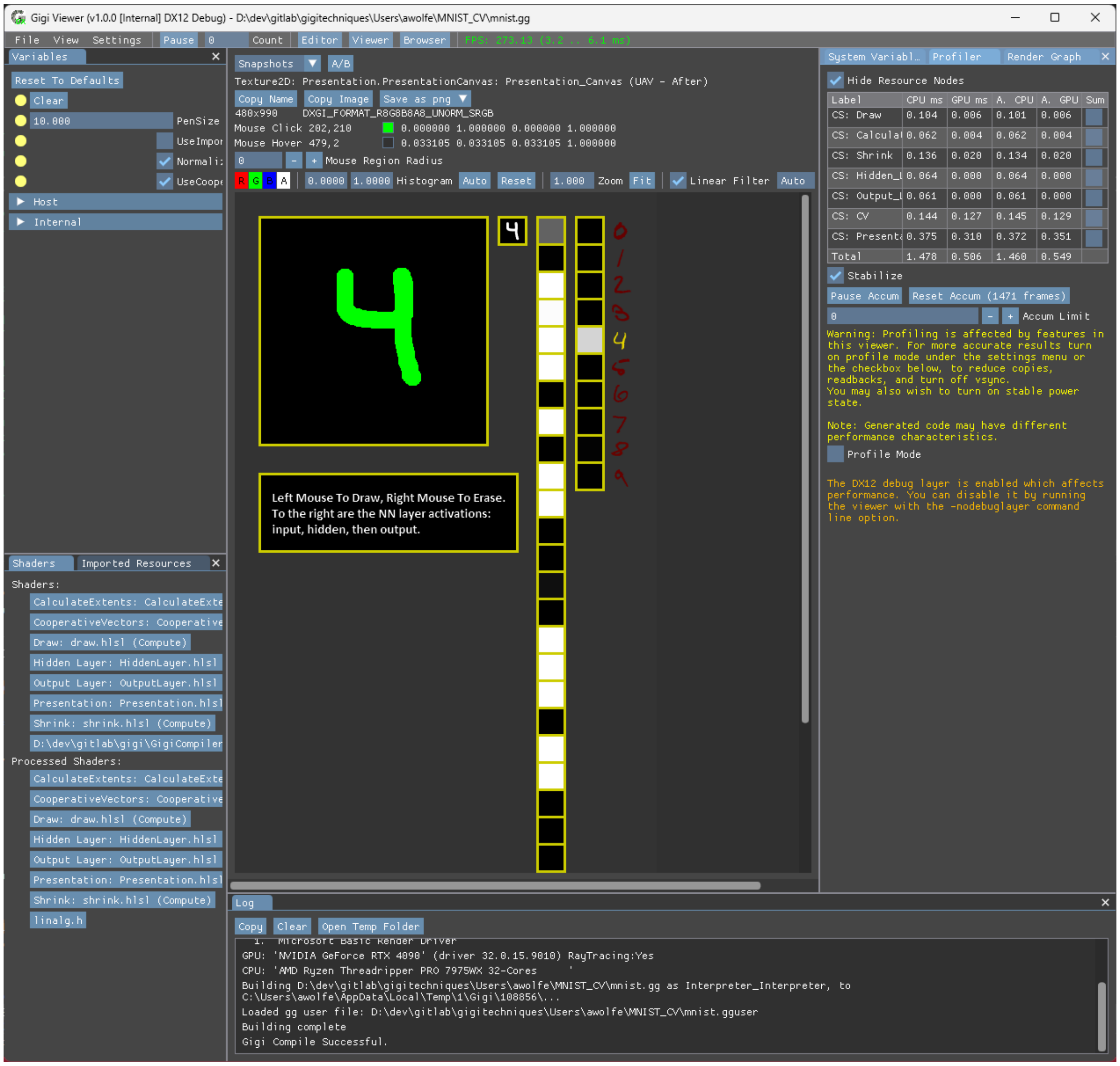
Working Demo in Gigi Browser – Cooperative Vector MNIST

In the Gigi Browser is a demo named “Cooperative Vector MNIST” that does real time inference on a numeric digit that you draw in a box. It has code paths for both cooperative vectors inference, as well as regular compute shaders.

This is a demo of functionality, but not a good demo of performance and actually slows down slightly when you use cooperative vectors instead of regular compute shaders. The reason for this is that the demo runs a neural network once per frame, while cooperative vectors are more designed to run once per pixel or similar. On my machine, the frame time is 0.140ms with compute shaders, and 0.155ms with cooperative vectors.

Assuming you did the setup work described earlier, after downloading the technique in the Gigi browser, you need to run the viewer with the command line parameters **-AgilitySDKPreview -nopixcapture -norenderdoc**. Then you can load the technique in the viewer.

Demo:



```
Gigi Viewer (v1.0.0 [Internal] DX12 Debug) - D:\dev\gitlab\gigitechniques\Users\awolfe\MNIST_CV\mnist.gg
FPS: 273.13 (3.2 .. 6.1 ms)

Texture2D: Presentation.PresentationCanvas: Presentation_Canvas (UAV - After)
480x990     DXGI_FORMAT_R8G8B8A8_UNORM_SRGB
Mouse Click 202,210      0.000000 1.000000 0.000000 1.000000
Mouse Hover 479,2        0.033105 0.033105 0.033105 1.000000
```

Left Mouse To Draw, Right Mouse To Erase. To the right are the NN layer activations: input, hidden, then output.

Label	CPU ms	GPU ms	A. CPU	A. GPU	Sum
CS: Draw	0.104	0.006	0.101	0.006	
CS: Calcula	0.062	0.004	0.062	0.004	
CS: Shrink	0.136	0.020	0.134	0.020	
CS: Hidden_L	0.064	0.000	0.064	0.000	
CS: Output_L	0.061	0.000	0.061	0.000	
CS: CV	0.144	0.127	0.145	0.129	
CS: Presenta	0.375	0.310	0.372	0.351	
Total	1.478	0.506	1.460	0.549	

Warning: Profiling is affected by features in this viewer. For more accurate results turn on profile mode under the settings menu or the checkbox below, to reduce copies, readbacks, and turn off vsync.
You may also wish to turn on stable power state.

Note: Generated code may have different performance characteristics.

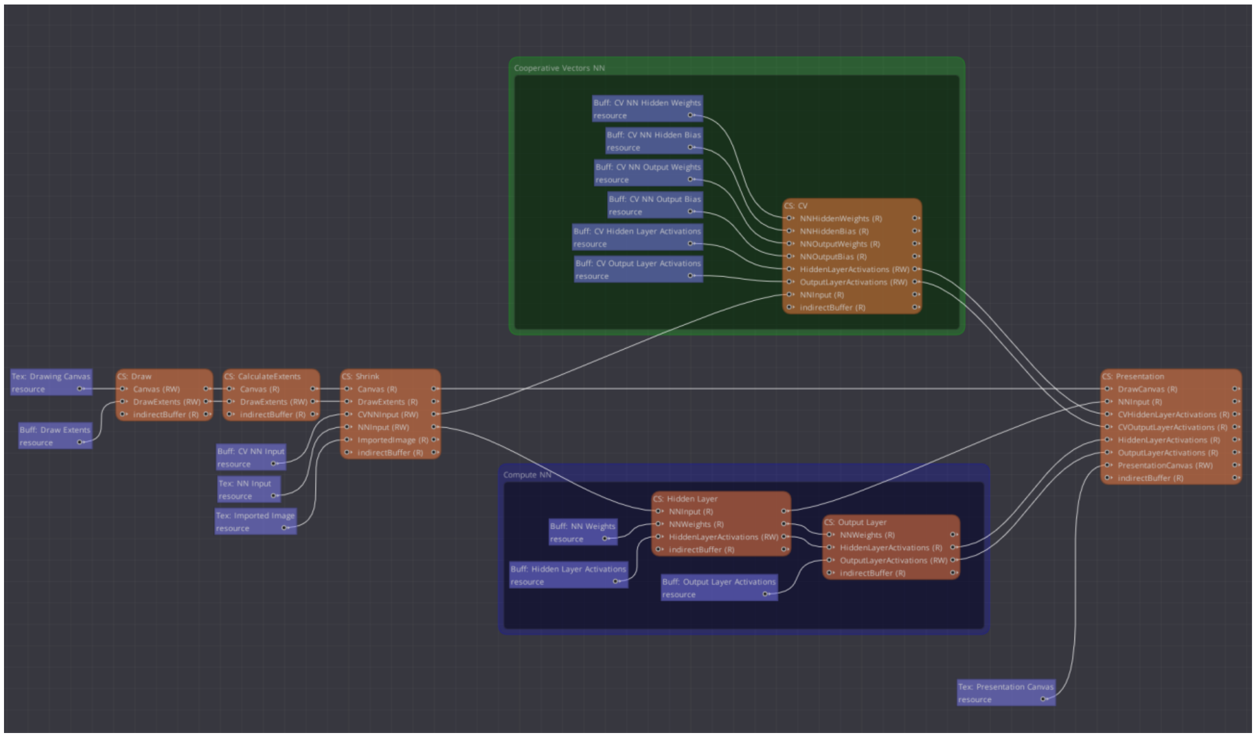
The DX12 debug layer is enabled which affects performance. You can disable it by running the viewer with the -nodebuglayer command line option.

Shaders:
- CalculateExtents: CalculateExte
- CooperativeVectors: Cooperative
- Draw: draw.hlsl (Compute)
- Hidden Layer: HiddenLayer.hlsl
- Output Layer: OutputLayer.hlsl
- Presentation: Presentation.hlsl
- Shrink: shrink.hlsl (Compute)
- D:\dev\gitlab\gigi\GigiCompiler

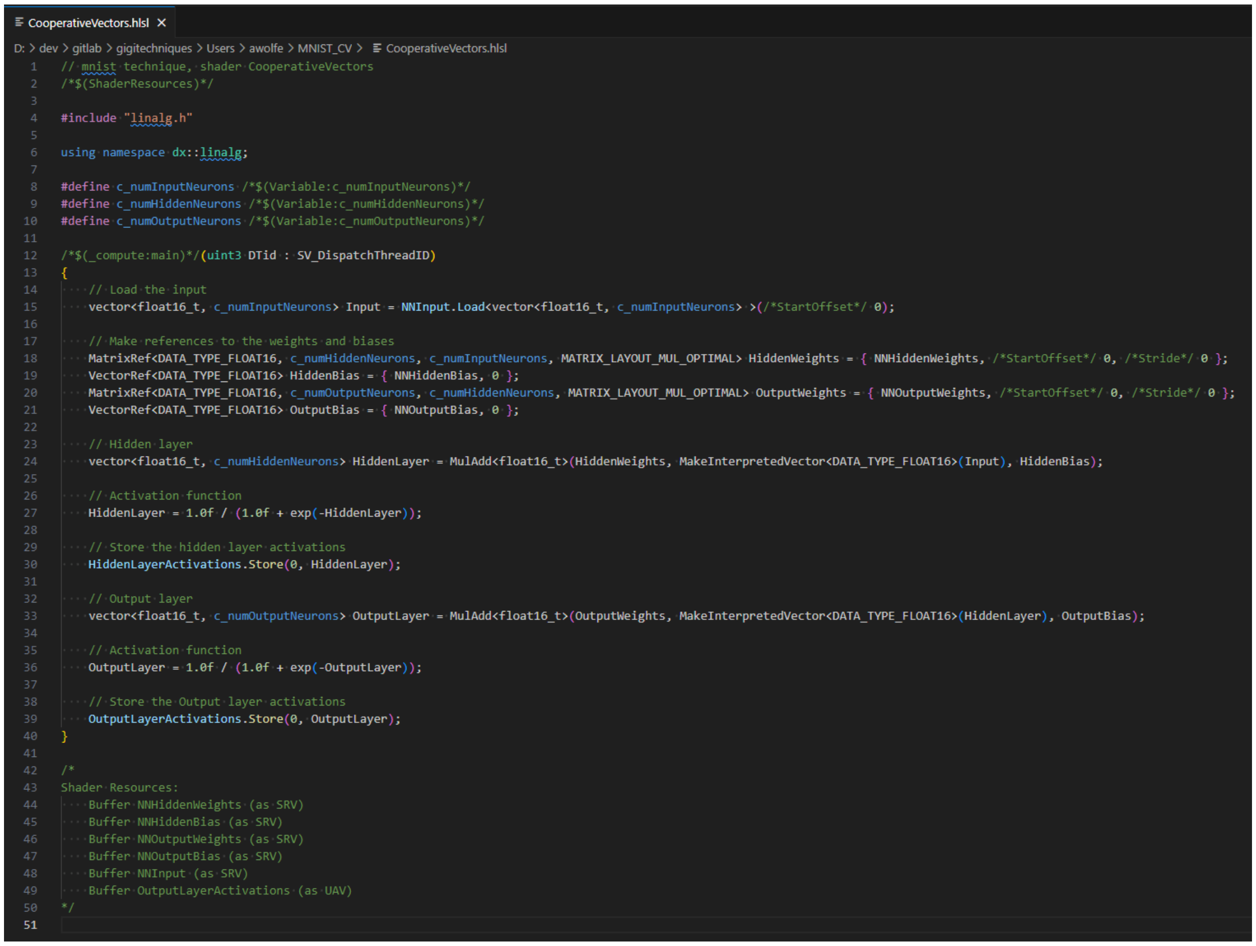
Processed Shaders:
- CalculateExtents: CalculateExte
- CooperativeVectors: Cooperative
- Draw: draw.hlsl (Compute)
- Hidden Layer: HiddenLayer.hlsl
- Output Layer: OutputLayer.hlsl
- Presentation: Presentation.hlsl
- Shrink: shrink.hlsl (Compute)
- linalg.h

```
  1.  Microsoft Basic Render Driver
GPU: 'NVIDIA GeForce RTX 4090' (driver 32.0.15.9010) RayTracing:Yes
CPU: 'AMD Ryzen Threadripper PRO 7975WX 32-Cores      '
Building D:\dev\gitlab\gigitechniques\Users\awolfe\MNIST_CV\mnist.gg as Interpreter_Interpreter, to
C:\Users\awolfe\AppData\Local\Temp\1\Gigi\108856\...
Loaded gg user file: D:\dev\gitlab\gigitechniques\Users\awolfe\MNIST_CV\mnist.gguser
Building complete
Gigi Compile Successful.
```

Render Graph:



Gigi Shader:

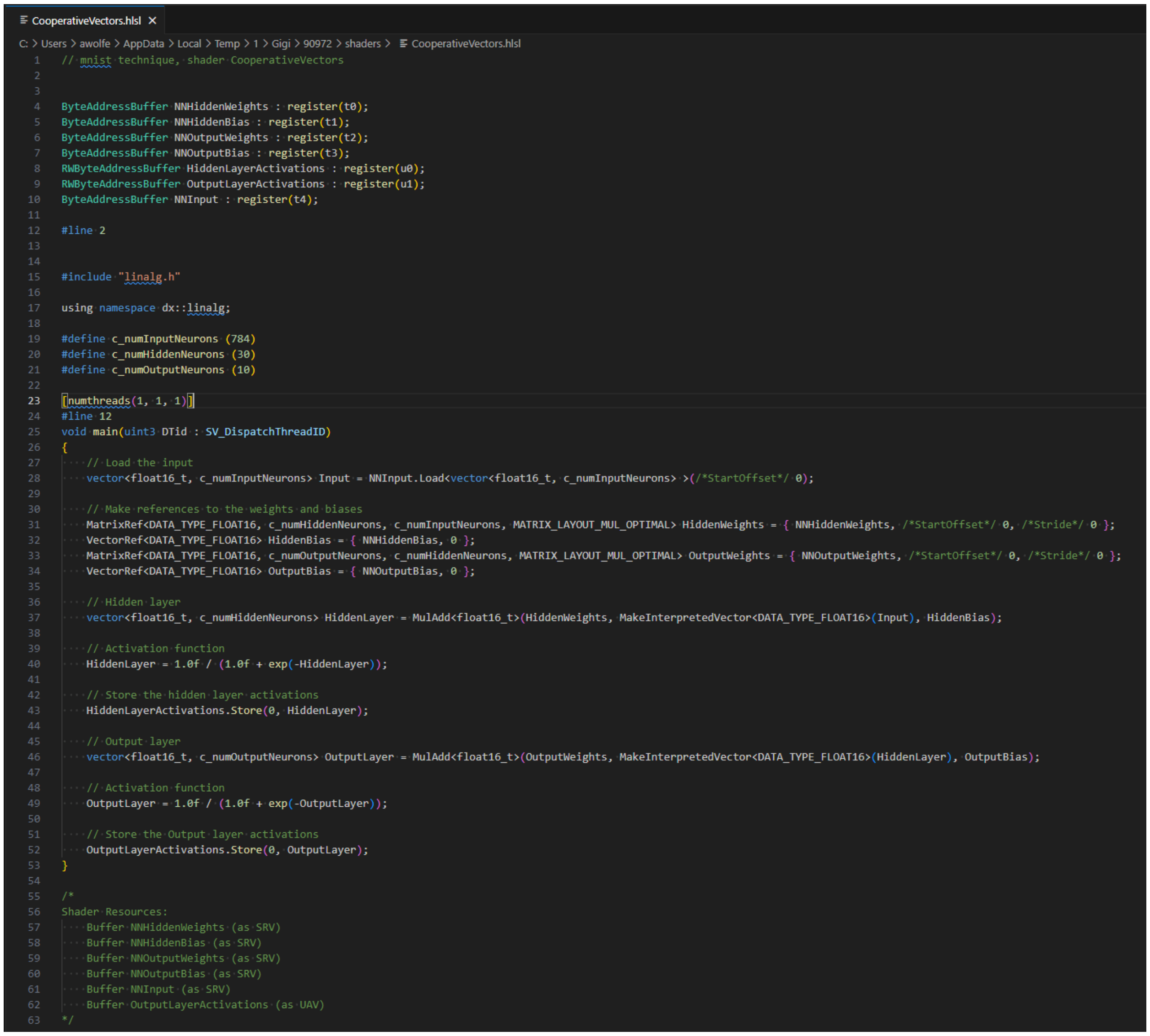


CooperativeVectors.hlsl

D: > dev > gitlab > gigitechniques > Users > awolfe > MNIST_CV > CooperativeVectors.hlsl

```hlsl
// mnist technique, shader CooperativeVectors
/*$(ShaderResources)*/

#include "linalg.h"

using namespace dx::linalg;

#define c_numInputNeurons /*$(Variable:c_numInputNeurons)*/
#define c_numHiddenNeurons /*$(Variable:c_numHiddenNeurons)*/
#define c_numOutputNeurons /*$(Variable:c_numOutputNeurons)*/

/*$(_compute:main)*/(uint3 DTid : SV_DispatchThreadID)
{
    // Load the input
    vector<float16_t, c_numInputNeurons> Input = NNInput.Load<vector<float16_t, c_numInputNeurons> >(/*StartOffset*/ 0);

    // Make references to the weights and biases
    MatrixRef<DATA_TYPE_FLOAT16, c_numHiddenNeurons, c_numInputNeurons, MATRIX_LAYOUT_MUL_OPTIMAL> HiddenWeights = { NNHiddenWeights, /*StartOffset*/ 0, /*Stride*/ 0 };
    VectorRef<DATA_TYPE_FLOAT16> HiddenBias = { NNHiddenBias, 0 };
    MatrixRef<DATA_TYPE_FLOAT16, c_numOutputNeurons, c_numHiddenNeurons, MATRIX_LAYOUT_MUL_OPTIMAL> OutputWeights = { NNOutputWeights, /*StartOffset*/ 0, /*Stride*/ 0 };
    VectorRef<DATA_TYPE_FLOAT16> OutputBias = { NNOutputBias, 0 };

    // Hidden layer
    vector<float16_t, c_numHiddenNeurons> HiddenLayer = MulAdd<float16_t>(HiddenWeights, MakeInterpretedVector<DATA_TYPE_FLOAT16>(Input), HiddenBias);

    // Activation function
    HiddenLayer = 1.0f / (1.0f + exp(-HiddenLayer));

    // Store the hidden layer activations
    HiddenLayerActivations.Store(0, HiddenLayer);

    // Output layer
    vector<float16_t, c_numOutputNeurons> OutputLayer = MulAdd<float16_t>(OutputWeights, MakeInterpretedVector<DATA_TYPE_FLOAT16>(HiddenLayer), OutputBias);

    // Activation function
    OutputLayer = 1.0f / (1.0f + exp(-OutputLayer));

    // Store the Output layer activations
    OutputLayerActivations.Store(0, OutputLayer);
}

/*
Shader Resources:
    Buffer NNHiddenWeights (as SRV)
    Buffer NNHiddenBias (as SRV)
    Buffer NNOutputWeights (as SRV)
    Buffer NNOutputBias (as SRV)
    Buffer NNInput (as SRV)
    Buffer OutputLayerActivations (as UAV)
*/

```

Processed Shader:


CooperativeVectors.hlsl
C: > Users > awolfe > AppData > Local > Temp > 1 > Gigi > 90972 > shaders > CooperativeVectors.hlsl

```hlsl
// mnist technique, shader CooperativeVectors


ByteAddressBuffer NNHiddenWeights : register(t0);
ByteAddressBuffer NNHiddenBias : register(t1);
ByteAddressBuffer NNOutputWeights : register(t2);
ByteAddressBuffer NNOutputBias : register(t3);
RWByteAddressBuffer HiddenLayerActivations : register(u0);
RWByteAddressBuffer OutputLayerActivations : register(u1);
ByteAddressBuffer NNInput : register(t4);

#line 2


#include "linalg.h"

using namespace dx::linalg;

#define c_numInputNeurons (784)
#define c_numHiddenNeurons (30)
#define c_numOutputNeurons (10)

[numthreads(1, 1, 1)]
#line 12
void main(uint3 DTid : SV_DispatchThreadID)
{
    // Load the input
    vector<float16_t, c_numInputNeurons> Input = NNInput.Load<vector<float16_t, c_numInputNeurons> >(/*StartOffset*/ 0);

    // Make references to the weights and biases
    MatrixRef<DATA_TYPE_FLOAT16, c_numHiddenNeurons, c_numInputNeurons, MATRIX_LAYOUT_MUL_OPTIMAL> HiddenWeights = { NNHiddenWeights, /*StartOffset*/ 0, /*Stride*/ 0 };
    VectorRef<DATA_TYPE_FLOAT16> HiddenBias = { NNHiddenBias, 0 };
    MatrixRef<DATA_TYPE_FLOAT16, c_numOutputNeurons, c_numHiddenNeurons, MATRIX_LAYOUT_MUL_OPTIMAL> OutputWeights = { NNOutputWeights, /*StartOffset*/ 0, /*Stride*/ 0 };
    VectorRef<DATA_TYPE_FLOAT16> OutputBias = { NNOutputBias, 0 };

    // Hidden layer
    vector<float16_t, c_numHiddenNeurons> HiddenLayer = MulAdd<float16_t>(HiddenWeights, MakeInterpretedVector<DATA_TYPE_FLOAT16>(Input), HiddenBias);

    // Activation function
    HiddenLayer = 1.0f / (1.0f + exp(-HiddenLayer));

    // Store the hidden layer activations
    HiddenLayerActivations.Store(0, HiddenLayer);

    // Output layer
    vector<float16_t, c_numOutputNeurons> OutputLayer = MulAdd<float16_t>(OutputWeights, MakeInterpretedVector<DATA_TYPE_FLOAT16>(HiddenLayer), OutputBias);

    // Activation function
    OutputLayer = 1.0f / (1.0f + exp(-OutputLayer));

    // Store the Output layer activations
    OutputLayerActivations.Store(0, OutputLayer);
}

/*
Shader Resources:
    Buffer NNHiddenWeights (as SRV)
    Buffer NNHiddenBias (as SRV)
    Buffer NNOutputWeights (as SRV)
    Buffer NNOutputBias (as SRV)
    Buffer NNInput (as SRV)
    Buffer OutputLayerActivations (as UAV)
*/
```