The APEX-DNN User Manual

Version	Details of Change	Author	Date
1.0	Initial Revision	Changyun (Nathan) Zhu	December 29, 2016
1.1	RTM 1.0 Revision	Changyun (Nathan) Zhu	June 21, 2017

Contents

Chapter 1

APEXDNN_SN Library

The APEXDNN_SN library provides basic functionality for developers to design their own squeezenet-liked convolution neural network based applications while taking advantage of NXP's massively parallel APEX architecture. The library contains the following SqueezeNet style modules as well as basic tensor operations.

Chapter 2

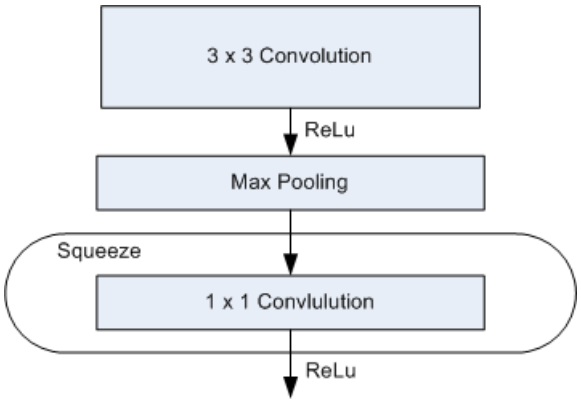
CONV3X3MPS1 CONV3X3MPS1 Module

As following figure shows it includes 3x3 convolution filter, followed by max pooling filter then squeeze 1x1 convolution filter. Following parameters can be configured when the module is created:

– 3x3 convolution filter's number of output channels; number of input channels; veritical and horizontal padding, striding.

– Max pooling filter's window size; vertical and horizontal padding and striding.

– 1x1 convolution filter's number of output channels; vertical and horizontal striding.



Convlution 3x3	Padding	0, 1
	Striding	1, 2
	Input Channels	1, 3,
	Output Channels	Multiple of 4
Max Pooling 3x3	Padding	0, 1
	Striding	1, 2, 4,...
	Output Channels	Multiple of 4
Squeeze 1x1	Padding	0
	Striding	1, 2, 4,...
	Output Channels	Any
NOTE:	Output width time number of output channels cannot be greater than 128 x 64;	

Table 2.1: CONV3X3MPS1 Module Configurations

Input Resolution		<227, 227>, <19, 39>, <29, 61> (Horizontal, Vertical)
Expand 3x3	Padding	<0, 0>, <1, 1> (Horizontal, Vertical)
	Striding	<1, 1>, <2, 2> (Horizontal, Vertical)
	Input Channels	1, 3
	Output Channels	64
Max Pooling 3x3	Padding	<0, 0>, <1, 1> (Horizontal, Vertical)
	Striding	<2, 2> (Horizontal, Vertical)
	Output Channels	64
Squeeze 1x1	Padding	<0, 0> (Horizontal, Vertical)
	Striding	<1, 1> (Horizontal, Vertical)
	Input Channels	64
	Output Channels	16

Table 2.2: CONV3X3MPS1 Module Tested Configurations

Chapter 3

E1E3MPS1 E1E3MPS1 Module

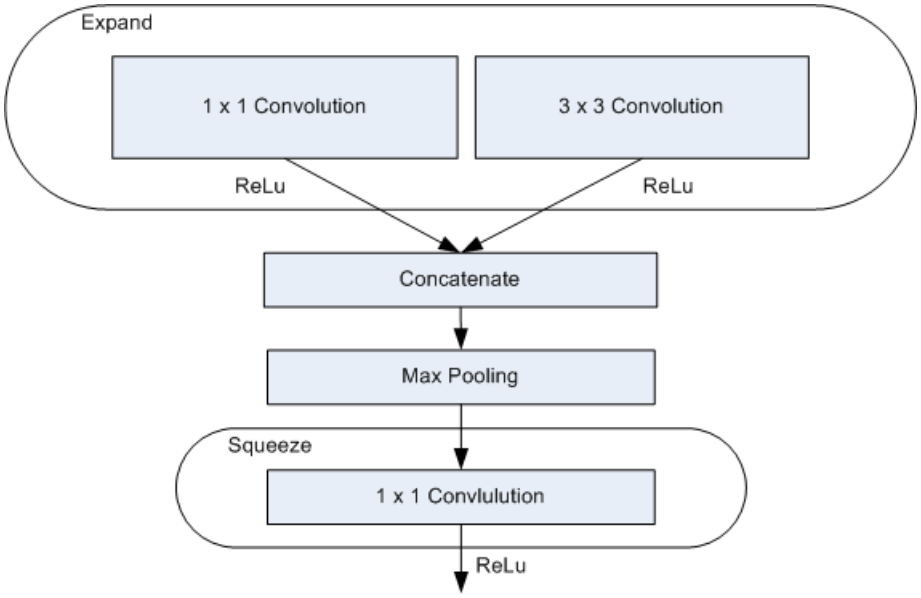
As following figure shows it includes expand 1x1 convolution filter and expand 3x3 convolution filter followed by max pooling filter then squeeze 1x1 convolution filter. Following parameters can be configured when the module is created:

– expand 1x1 convolution filter's number of output channels; number of input channels; veritical and horizontal striding.

– expand 3x3 convolution filter's number of output channels; number of input channels; veritical and horizontal padding, striding.

– Maxpooling filter's window size; vertical and horizontal padding, striding.

– 1x1 convolution filter's number of output channels; vertical and horizontal striding.



Expand 1x1	Padding	0
	Striding	1, 2, 4, ...
	Input Channels	Multiple of 4
	Output Channels	Multiple of 4
Expand 3x3	Padding	1
	Striding	1, 2, 4, ...
	Input Channels	Multiple of 4
	Output Channels	Multiple of 4
Max Pooling 3x3	Padding	0, 1
	Striding	1, 2, 4,...
	Output Channels	Multiple of 4
Squeeze 1x1	Padding	0
	Striding	1, 2, 4,...
	Output Channels	Any
NOTE:	Only support symmetric Expands, i.e. Expand 1x1 and Expand 3x3 must have equal number of output channels	
	Only support zero padding	
	Output width time number of output channels cannot be greater than 128 x 64;	

Table 3.1: E1E3MPS1 Module Configurations

Input Resolution		<56, 56> (Horizontal, Vertical)
Expand 1x1	Padding	<0, 0> (Horizontal, Vertical)
	Striding	<1, 1> (Horizontal, Vertical)
	Input Channels	16
	Output Channels	64
Expand 3x3	Padding	<1, 1> (Horizontal, Vertical)
	Striding	<1, 1> (Horizontal, Vertical)
	Input Channels	16
	Output Channels	64
Max Pooling 3x3	Padding	<0, 0> (Horizontal, Vertical)
	Striding	<2, 2> (Horizontal, Vertical)
	Output Channels	128
Squeeze 1x1	Padding	<0, 0> (Horizontal, Vertical)
	Striding	<1, 1> (Horizontal, Vertical)
	Input Channels	128
	Output Channels	32

Table 3.2: E1E3MPS1 Module Tested Configurations

Chapter 4

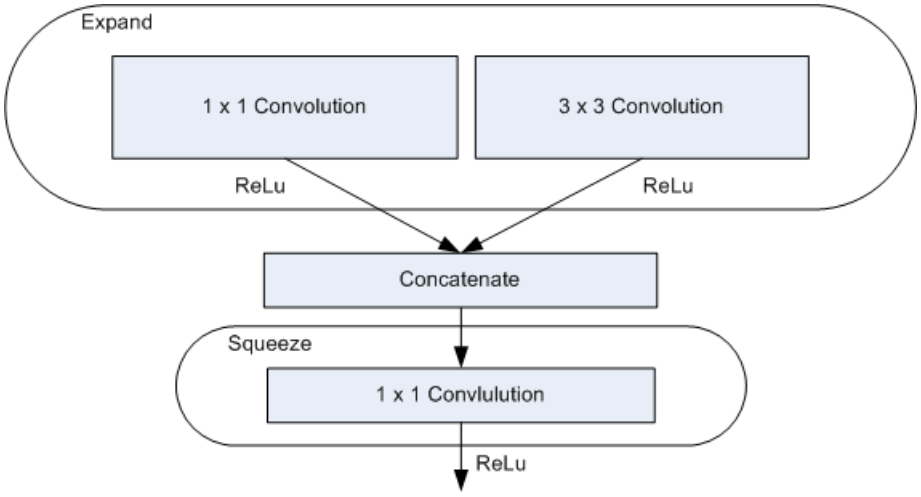
E1E3S1 Module

As following figure shows it includes expand 1x1 convolution filter and expand 3x3 convolution filter followed by squeeze 1x1 convolution filter. Following parameters can be configured when the module is created:

– expand 1x1 convolution filter's number of output channels; number of input channels; veritical and horizontal striding.

– expand 3x3 convolution filter's number of output channels; number of input channels; veritical and horizontal padding, striding.

– 1x1 convolution filter's number of output channels; vertical and horizontal striding.



Expand 1x1	Padding	0
	Striding	1, 2, 4, ...
	Input Channels	Multiple of 4
	Output Channels	Multiple of 4
Expand 3x3	Padding	1
	Striding	1, 2, 4, ...
	Input Channels	Multiple of 4
	Output Channels	Multiple of 4
Squeeze 1x1	Padding	0
	Striding	1, 2, 4,...
	Input Channels	Multiple of 4
	Output Channels	Any
NOTE:	Only support symmetric Expands, i.e. Expand 1x1 and Expand 3x3 must have equal number of output channels	
	Only support zero padding	
	Output width time number of output channels cannot be greater than 128 x 64;	

Table 4.1: E1E3S1 Module Configurations

Input Resolution		<56, 56> (Horizontal, Vertical)
Input Channels		16
Expand 1x1	Padding	<0, 0> (Horizontal, Vertical)
	Striding	<1, 1> (Horizontal, Vertical)
	Input Channels	16
	Output Channels	64
Expand 3x3	Padding	<1, 1> (Horizontal, Vertical)
	Striding	<1, 1> (Horizontal, Vertical)
	Input Channels	16
	Output Channels	64
Squeeze 1x1	Padding	<0, 0> (Horizontal, Vertical)
	Striding	<1, 1> (Horizontal, Vertical)
	Input Channels	128
	Output Channels	16

Table 4.2: E1E3S1 Module Tested Configurations

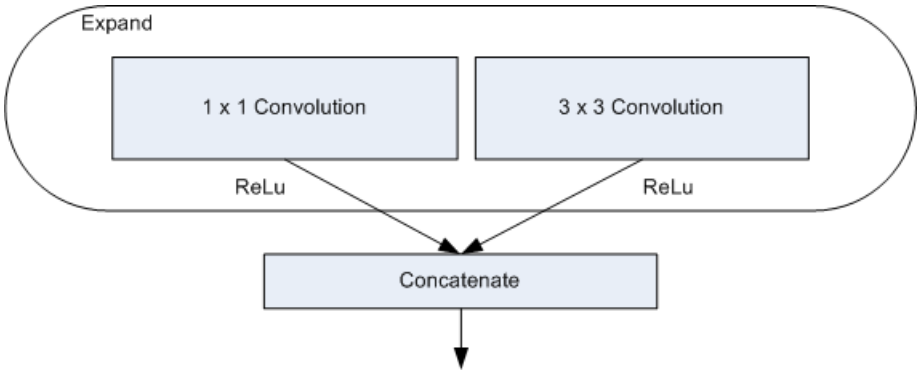
Chapter 5

E1E3 Module

As following figure shows it includes expand 1x1 convolution filter and expand 3x3 convolution filter. Following parameters can be configured when the module is created:

– expand 1x1 convolution filter's number of output channels; number of input channels; veritical and horizontal striding.

– expand 3x3 convolution filter's number of output channels; number of input channels; veritical and horizontal padding, striding.



Expand 1x1	Padding	0
	Striding	1, 2, 4, ...
	Input Channels	Multiple of 4
	Output Channels	Any
Expand 3x3	Padding	1
	Striding	1, 2, 4, ...
	Input Channels	Multiple of 4
	Output Channels	Any
NOTE:	Only support symmetric Expands, i.e. Expand 1x1 and Expand 3x3 must have equal number of output channels	
	Only support zero padding	
	Output width time number of output channels cannot be greater than 128 x 64;	

Table 5.1: E1E3 Module Configurations

Input Resolution		<14, 14> (Horizontal, Vertical)
Input Channels		64
Expand 1x1	Padding	<0, 0> (Horizontal, Vertical)
	Striding	<1, 1> (Horizontal, Vertical)
	Output Channels	256
Expand 3x3	Padding	<1, 1> (Horizontal, Vertical)
	Striding	<1, 1> (Horizontal, Vertical)
	Output Channels	256

Table 5.2: E1E3 Module Tested Configurations

Chapter 6

Implementing SqueezeNet V1.1 using APEX-DNN Modules

Following figure shows how to use APEX-DNN modules to build up SqueezeNet V1.1. Left column shows regular layers in the network. For limited space, we ignore all the ReLu layers. Middle columns shows how to merge and map those layers to APEX-DNN optimized modules implementation and right column give the brief summary what filters are included in each module.

Squeeze Net V1.1 layers	APEX DNN Modules	Module Functionality
conv1 pool1 fire2/squeeze1x1	CONV3X3MPS1	Convolution 3x3 -> Max Pooling 3 x 3-> Squeeze 1x1
fire2/expand1x1 "fire2/expand3x3" fire2/concat fire3/squeeze1x1	E1E3S1	Expand 1x1 + Expand 3x3 -> Concatenate -> Squeeze 1x1
fire3/expand1x1 "fire3/expand3x3" fire3/concat pool3 fire4/squeeze1x1	E1E3MPS1	Expand 1x1 + Expand 3x3 -> Concatenate -> Max Pooling 3xx3 -> Squeeze 1x1
fire4/expand1x1 "fire4/expand3x3" fire4/concat fire5/squeeze1x1	E1E3S1	Expand 1x1 + Expand 3x3 -> Concatenate -> Squeeze 1x1
fire5/expand1x1 "fire5/expand3x3" fire5/concat pool5 fire6/squeeze1x1	E1E3MPS1	Expand 1x1 + Expand 3x3 -> Concatenate -> Max Pooling 3xx3 -> Squeeze 1x1
fire6/expand1x1 "fire6/expand3x3" fire6/concat fire7/squeeze1x1	E1E3S1	Expand 1x1 + Expand 3x3 -> Concatenate -> Squeeze 1x1
fire7/expand1x1 "fire7/expand3x3" fire7/concat fire8/squeeze1x1	E1E3S1	Expand 1x1 + Expand 3x3 -> Concatenate -> Squeeze 1x1
fire8/expand1x1 "fire8/expand3x3" fire8/concat fire9/squeeze1x1	E1E3S1	Expand 1x1 + Expand 3x3 -> Concatenate -> Squeeze 1x1
fire9/expand1x1 "fire9/expand3x3" fire9/concat	E1E3	Expand 1x1 + Expand 3x3 -> Concatenate
conv10 pool10	E1AP	Expand 1x1 + Sum Pooling

Chapter 7

APEX-DNN Example use case

```
static void dump_4d_tensor_desc(apexdnnTensorDescriptor* TensorDesc, char* filename, apexdnnTensorFormat_t
      format)
{
#ifndef __STANDALONE__
   FILE *fp;
   fp = fopen(filename, "wb");
#else
   int fp = 0;
   fp = T32_fopen(filename, T32_TERM_O_CREATE_TRUNC | T32_TERM_O_RDWR | T32_TERM_O_BINARY);
#endif
   if (format != APEXDNN_TENSOR_NHCW)
   {
      printf("ERROR: Unsupported dump format!\n");
      return;
   }

   int map[4];
   int coordinate[4];
   int N = 0;
   int H = 0;
   int C = 0;
   int W = 0;

   if (apexdnnRetTensorFormat(TensorDesc) && format == APEXDNN_TENSOR_NHCW)
   {
      N = apexdnnRetTensorDim(TensorDesc, 0);
      H = apexdnnRetTensorDim(TensorDesc, 1);
      C = apexdnnRetTensorDim(TensorDesc, 2);
      W = apexdnnRetTensorDim(TensorDesc, 3);
      map[0] = 0;
      map[1] = 1;
      map[2] = 2;
      map[3] = 3;
   }
   else if (apexdnnRetTensorFormat(TensorDesc) == APEXDNN_TENSOR_NCHW && format ==
      APEXDNN_TENSOR_NHCW)
   {
      N = apexdnnRetTensorDim(TensorDesc, 0);
      H = apexdnnRetTensorDim(TensorDesc, 2);
      C = apexdnnRetTensorDim(TensorDesc, 1);
      W = apexdnnRetTensorDim(TensorDesc, 3);
      map[0] = 0;
      map[1] = 2;
      map[2] = 1;
      map[3] = 3;
   }
   else
   {
      printf("ERROR: Unsupported dump format!\n");
      return;
   }

   char resstr[512];
   for (int d0 = 0; d0 < N; d0++)
   {
      for (int d1 = 0; d1 < H; d1++)
      {
```

```
         for (int d2 = 0; d2 < C; d2++)
         {
            for (int d3 = 0; d3 < W; d3++)
            {
               coordinate[map[0]] = d0;
               coordinate[map[1]] = d1;
               coordinate[map[2]] = d2;
               coordinate[map[3]] = d3;
               if (apexdnnRetTensorDataType(TensorDesc) == APEXDNN_DATA_8BIT)
               {
                  int8_t *p = (int8_t*)apexdnnRetTensorDataPtr(TensorDesc);
                  sprintf(resstr, "%d, ", *(p + coordinate[0] * apexdnnRetTensorStride(TensorDesc, 0)
                                            + coordinate[1] * apexdnnRetTensorStride(TensorDesc, 1)
                                            + coordinate[2] * apexdnnRetTensorStride(TensorDesc, 2)
                                            + coordinate[3] * apexdnnRetTensorStride(TensorDesc, 3)));
#ifndef __STANDALONE__
                  fwrite(resstr, strlen(resstr), 1, fp);
#else
                  T32_fwrite(resstr, strlen(resstr), 1, fp);
#endif
               }
               else if (apexdnnRetTensorDataType(TensorDesc) ==
      APEXDNN_DATA_16BIT)
               {
                  int16_t *p = (int16_t*)apexdnnRetTensorDataPtr(TensorDesc);
                  sprintf(resstr, "%d, ", *(p + coordinate[0] * apexdnnRetTensorStride(TensorDesc, 0)
                                            + coordinate[1] * apexdnnRetTensorStride(TensorDesc, 1)
                                            + coordinate[2] * apexdnnRetTensorStride(TensorDesc, 2)
                                            + coordinate[3] * apexdnnRetTensorStride(TensorDesc, 3)));
#ifndef __STANDALONE__
                  fwrite(resstr, strlen(resstr), 1, fp);
#else
                  T32_fwrite(resstr, strlen(resstr), 1, fp);
#endif
               }
            }
         }
         sprintf(resstr, "\n");
#ifndef __STANDALONE__
         fwrite(resstr, strlen(resstr), 1, fp);
#else
         T32_fwrite(resstr, strlen(resstr), 1, fp);
#endif
      }
   }
#ifndef __STANDALONE__
   fclose(fp);
#else
   T32_fclose(fp);
#endif
   return;
}

/*!*****************************************************************************
 * \brief APEX-DNN test case 1.

 * This is building SqueezeNet V1.1 up to FIRE9/CONCAT layer as a feature extractor.
 * Network contains following filters:

 * CONV1
 * MAXPOOLING 1
 * FIRE2/SQUEEZE1X1

 * FIRE2/EXPAND1X1 FIRE2/EXPAND3X3
 * FIRE3/SQUEEZE1X1

 * FIRE3/EXPAND1X1 FIRE3/EXPAND3X3
 * MAXPOOLING 3
 * FIRE4/SQUEEZE1X1

 * FIRE4/EXPAND1X1 FIRE4/EXPAND3X3
 * FIRE5/SQUEEZE1X1

 * FIRE5/EXPAND1X1 FIRE5/EXPAND3X3
 * MAXPOOLING 5
 * FIRE6/SQUEEZE1X1

 * FIRE6/EXPAND1X1 FIRE6/EXPAND3X3
 * FIRE7/SQUEEZE1X1

 * FIRE7/EXPAND1X1 FIRE7/EXPAND3X3
```

```
 * FIRE8/SUQEEZE1X1

 * FIRE8/EXPAND1X1 FIRE8/EXPAND3X3
 * FIRE9/SQUEEZE1X1

 * FIRE9/EXPAND1X1 FIRE9/EXPAND3X3
                                                                                */
static void build_sn11_fire9(apexdnnNet *Net)
{
   void* Layer;

   /*                          0   1   2   3   4    5   6   7    8    9   10   11   12   13   14   15   16
        17   18   19   20   21   22   23   24   25   26   27  */
   int INPUT_CHANNELS[28]  = { 3, 64, 64, 16, 16, 128, 16, 16, 128, 128,  32,  32, 256,  32,  32, 256, 256,
         48,  48, 384,  48,  48, 384,  64,  64, 512,  64,  64};
   int OUTPUT_CHANNELS[28] = {64, 64, 16, 64, 64,  16, 64, 64, 128,  32, 128, 128,  32, 128, 128, 256,  48,
       192, 192,  48, 192, 192,  64, 256, 256,  64, 256, 256};
   int H_WINDOW_SIZE[28]   = { 3,  3,  1,  1,  3,  1,  1,  3,   3,   1,   1,   3,   1,   1,   3,   3,   1,
         1,   3,   1,   1,   3,   1,   1,   3,   1,   1,   3};
   int W_WINDOW_SIZE[28]   = { 3,  3,  1,  1,  3,  1,  1,  3,   3,   1,   1,   3,   1,   1,   3,   3,   1,
         1,   3,   1,   1,   3,   1,   1,   3,   1,   1,   3};
   int H_PAD[28]           = { 0,  0,  0,  0,  1,  0,  0,  1,   0,   0,   0,   1,   0,   0,   1,   0,   0,
         0,   1,   0,   0,   1,   0,   0,   1,   0,   0,   1};
   int W_PAD[28]           = { 0,  0,  0,  0,  1,  0,  0,  1,   0,   0,   0,   1,   0,   0,   1,   0,   0,
         0,   1,   0,   0,   1,   0,   0,   1,   0,   0,   1};
   int H_STRIDE[28]        = { 2,  2,  1,  1,  1,  1,  1,  1,   2,   1,   1,   1,   1,   1,   1,   2,   1,
         1,   1,   1,   1,   1,   1,   1,   1,   1,   1,   1};
   int W_STRIDE[28]        = { 2,  2,  1,  1,  1,  1,  1,  1,   2,   1,   1,   1,   1,   1,   1,   2,   1,
         1,   1,   1,   1,   1,   1,   1,   1,   1,   1,   1};

   int MODULE_FILTER_START_IDX[9] = {0, 3, 6, 10, 13, 17, 20, 23, 26};

   for (int module = 0; module < 9; module++)
   {
      int FilterIdx = MODULE_FILTER_START_IDX[module];
      if (module == 0)
      {
         apexdnnCreateConv3x3MPS1Module(
               ((apexdnnConv3x3MPS1Module**)&Layer),
               APEXDNN_DATA_8BIT,
               OUTPUT_CHANNELS[FilterIdx], INPUT_CHANNELS[FilterIdx], H_PAD[FilterIdx], W_PAD[FilterIdx],
      H_STRIDE[FilterIdx], W_STRIDE[FilterIdx],
               H_WINDOW_SIZE[FilterIdx+1], W_WINDOW_SIZE[FilterIdx+1], H_PAD[FilterIdx+1], W_PAD[FilterIdx+
      1], H_STRIDE[FilterIdx+1], W_STRIDE[FilterIdx+1],
               OUTPUT_CHANNELS[FilterIdx+2], H_STRIDE[FilterIdx+2], W_STRIDE[FilterIdx+2]);
      }
      else if (module == 1 || module == 3 || module == 5 || module == 6 || module == 7)
      {
         apexdnnCreateE1E3S1Module(
               ((apexdnnE1E3S1Module**)&Layer),
               APEXDNN_DATA_8BIT,
               OUTPUT_CHANNELS[FilterIdx], INPUT_CHANNELS[FilterIdx], H_STRIDE[FilterIdx], W_STRIDE[
      FilterIdx],
               OUTPUT_CHANNELS[FilterIdx+1], INPUT_CHANNELS[FilterIdx+1], H_PAD[FilterIdx+1], W_PAD[
      FilterIdx+1], H_STRIDE[FilterIdx+1], W_STRIDE[FilterIdx+1],
               OUTPUT_CHANNELS[FilterIdx+2], H_STRIDE[FilterIdx+2], W_STRIDE[FilterIdx+2]);
      }
      else if (module == 2 || module == 4)
      {
         apexdnnCreateE1E3MPS1Module(
               ((apexdnnE1E3MPS1Module**)&Layer),
               APEXDNN_DATA_8BIT,
               OUTPUT_CHANNELS[FilterIdx], INPUT_CHANNELS[FilterIdx], H_STRIDE[FilterIdx], W_STRIDE[
      FilterIdx],
               OUTPUT_CHANNELS[FilterIdx+1], INPUT_CHANNELS[FilterIdx+1], H_PAD[FilterIdx+1], W_PAD[
      FilterIdx+1], H_STRIDE[FilterIdx+1], W_STRIDE[FilterIdx+1],
               H_WINDOW_SIZE[FilterIdx+2], W_WINDOW_SIZE[FilterIdx+2], H_PAD[FilterIdx+2], W_PAD[FilterIdx+
      2], H_STRIDE[FilterIdx+2], W_STRIDE[FilterIdx+2],
               OUTPUT_CHANNELS[FilterIdx+3], H_STRIDE[FilterIdx+3], W_STRIDE[FilterIdx+3]);
      }
      else if (module == 8)
      {
         apexdnnCreateE1E3Module(
               ((apexdnnE1E3Module**)&Layer),
               APEXDNN_DATA_8BIT,
               OUTPUT_CHANNELS[FilterIdx], INPUT_CHANNELS[FilterIdx], H_STRIDE[FilterIdx], W_STRIDE[
      FilterIdx],
               OUTPUT_CHANNELS[FilterIdx+1], INPUT_CHANNELS[FilterIdx+1], H_PAD[FilterIdx+1], W_PAD[
      FilterIdx+1], H_STRIDE[FilterIdx+1], W_STRIDE[FilterIdx+1]);
      }
      apexdnnNetAppendLayer(Net, (void*)Layer);
```

```
    }
}

static void build_case1_net(apexdnnNet *Net)
{
    void* Layer;

    build_sn11_fire9(Net);

    /* Average pooling is doing sum instead to keep more precision,
     * which make the output tensor is signed 16bit */
    apexdnnCreateE1APModule(
          ((apexdnnE1APModule**)&Layer),
          APEXDNN_DATA_8BIT,
          1000,
          512,
          1,
          1);

    apexdnnNetAppendLayer(Net, (void*)Layer);
}

int test_apexdnn_sn_case1(const char* src_path, const char* dst_path, const char* image)
{
    const int64_t cOalMemoryFreeSize_before = OAL_MemorySizeFree();

    int INPUT_WIDTH   = 227;
    int INPUT_HEIGHT  = 227;
    int INPUT_CHANNEL = 3;

    int reval = 0;
    char filename[256];
    FILE *fp;

    /* temporary buffer to read in model file */
    int8_t *ModelBuf;
    /* temporary tensor to read in input */
    apexdnnTensorDescriptor *TempTensor;

    /* workspace */
    apexdnnWorkSpace* Workspace;

    /* create workspace to:
     *    -- allocate intermediate buffer to avoid dynamic alloc/dealloc
     *    -- associate 64 CU of APEX 0 (the only supported case for now)
     */
    apexdnnCreateWorkSpace(&Workspace,
          ACF_APU_CFG__APU_0_CU_0_63_SMEM_0_3,
          0);

    /*
     * Read In Model file and store in temporary buffer
     */
#ifndef __STANDALONE__
    sprintf(filename, "%scase1/SqueezeV11Quant8_ILSVRC12.model", src_path);
#else
    sprintf(filename, "%scase1\\SqueezeV11Quant8_ILSVRC12.model", src_path);
#endif
    ModelBuf = (int8_t*)malloc(APEXCV_SQUEEZENET_V11_MODEL_BYTES);
    if (file_read_helper(ModelBuf, APEXCV_SQUEEZENET_V11_MODEL_BYTES, filename))
    {
        reval |= 1;
        return reval;
    }

    /*
     * Read Input and store in temporary compact N-C-H-W tensor
     */
    apexdnnCreate4dTensorDescriptor(
          &TempTensor,
          APEXDNN_DATA_8BIT,
          APEXDNN_TENSOR_NCHW,
          APEXDNN_TENSOR_MEM_HEAP,
          1, INPUT_CHANNEL, INPUT_HEIGHT, INPUT_WIDTH);
#ifndef __STANDALONE__
    sprintf(filename, "%scase1/case1_input_3x227x227.bin", src_path);
#else
    sprintf(filename, "%scase1\\case1_input_3x227x227.bin", src_path);
#endif
    if (file_read_helper(apexdnnRetTensorDataPtr(TempTensor), INPUT_CHANNEL *
        INPUT_HEIGHT * INPUT_WIDTH, filename))
```

```
    {
        reval |= 1;
        return reval;
    }

    /*
     * Build and run reference model network which will inference on host CPU
     */
    apexdnnNet*              Ref_Net            = NULL;
    apexdnnTensorDescriptor* Ref_NetInputTensor = NULL;
    apexdnnTensorDescriptor* Ref_NetOutputTensor = NULL;

    /*
     * Build empty reference model network
     */
    apexdnnCreateEmptyNet(&Ref_Net);
    build_case1_net(Ref_Net);

    /*
     * Fill reference networks's weight/bias tensors and quant parameters
     */
    apexdnnNetFillModel(Ref_Net, ModelBuf);

    /*
     * Create virtual tensor to represent input size.
     * Only need virtual tensor without allocating memory. The memory will be allocated
     * automatically when APEX-DNN verify the net based on which HW computing unit will be used
     * to inference the net to handle padding or make HW DMA optimal
     */
    apexdnnCreateVirtual4dTensorDescriptor(
          &Ref_NetInputTensor,
          APEXDNN_DATA_8BIT,
          APEXDNN_TENSOR_NCHW,
          APEXDNN_TENSOR_MEM_HEAP,
          1, INPUT_CHANNEL, INPUT_HEIGHT, INPUT_WIDTH);

    /*
     * Verify the network for host CPU inference. In this routine,
     *    -- intermediate tensor will be allocated
     *    -- Network input tensor will be re-organized and needed buffer, into which the input data needs
     *       to be feeded, will be allocated.
     *    -- Network output tensor will be created and associated memory will be allocated
     */
    apexdnnNetVerifyGraphCpu(Ref_Net, Ref_NetInputTensor, &Ref_NetOutputTensor);

    /*
     * Display network and input output tensor, if SHOW_NETWORK_STRUCTURE==1 for examination
     */
#if SHOW_NETWORK_STRUCTURE
    display_net(Ref_NetInputTensor, Ref_Net, Ref_NetOutputTensor);
#endif
    /*
     * Feed input by transforming temp tensor into network input tensor. Internally memory will be copied
     */
    apexdnnTransform4dTensorDescriptor(TempTensor, Ref_NetInputTensor);

    /*
     * Forware network calculation
     */
    apexdnnNetForwardCpu(Workspace, Ref_Net, Ref_NetInputTensor);

    /*
     * Dump each layer/module's output tensor into file for examination
     */
#ifndef __STANDALONE__
    for (int i = 0; i < apexdnnRetNetNumofLayers(Ref_Net); i++)
    {
        sprintf(filename, "%scase1_L%d_output_cmem_cpu.csv", dst_path, i);
        dump_4d_tensor_desc(apexdnnRetNetLayerOutputTensorDesc(Ref_Net, i),
         filename, APEXDNN_TENSOR_NHCW);
    }
#endif

    /*
     * Build and run network which will inference on APEX
     */
    apexdnnNet*              Apex_Net            = NULL;
    apexdnnTensorDescriptor* Apex_NetInputTensor = NULL;
    apexdnnTensorDescriptor* Apex_NetOutputTensor = NULL;

    /*
```

```
    * Build empty network
    */
   apexdnnCreateEmptyNet(&Apex_Net);
   build_case1_net(Apex_Net);

   /*
    * Fill networks's weight/bias tensors and quant parameters
    */
   apexdnnNetFillModel(Apex_Net, ModelBuf);

   SWT_ARM_LOG_MODULE_FILENAME_REG("apexdnn_sn_test", "test_apexdnn_sn_case1");

   /*
    * Create virtual tensor to represent input size.
    * Only need virtual tensor without allocating memory. The memory will be allocated
    * automatically when APEX-DNN verify the net based on which HW computing unit will be used
    * to inference the net to handle padding or make HW DMA optimal
    */
   apexdnnCreateVirtual4dTensorDescriptor(
         &Apex_NetInputTensor,
         APEXDNN_DATA_8BIT,
         APEXDNN_TENSOR_NCHW,
         APEXDNN_TENSOR_MEM_HEAP,
         1, INPUT_CHANNEL, INPUT_HEIGHT, INPUT_WIDTH);
   /*
    * Verify the network for host APU inference. In this routine,
    *    -- intermediate tensor will be allocated
    *    -- Network input tensor will be re-organized and needed buffer, into which the input data needs
    *       to be feeded, will be allocated.
    *    -- Network output tensor will be created and associated memory will be allocated
    */
   apexdnnNetVerifyGraphApex(Workspace, Apex_Net, Apex_NetInputTensor, &
      Apex_NetOutputTensor);
#if SHOW_NETWORK_STRUCTURE
   display_net(Apex_NetInputTensor, Apex_Net, Apex_NetOutputTensor);
#endif

   /*
    * Feed input by transforming temp tensor into network input tensor. Internally memory will be copied
    */
   apexdnnTransform4dTensorDescriptor(TempTensor, Apex_NetInputTensor);

   /*
    * Forware network calculation
    */
   SWT_ARM_LOG_IMAGE_SIZE_FUNCTION_REG(INPUT_WIDTH, INPUT_HEIGHT, 0, 0, 0, 0, 0, 0, 0, 0);
   reval |= apexdnnNetForwardApex(Workspace, Apex_Net, Apex_NetInputTensor, 0);
   SWT_ARM_LOG_FUNCTION_RVAL(reval);

   /*
    * Dump each layer/module's output tensor into file for examination
    */
#ifndef __STANDALONE__
   for ( int i = 0; i < apexdnnRetNetNumofLayers(Apex_Net); i++)
   {
      sprintf(filename, "%scase1_L%d_output_cmem_apex.csv", dst_path, i);
      dump_4d_tensor_desc(apexdnnRetNetLayerOutputTensorDesc(Apex_Net, i)
      , filename, APEXDNN_TENSOR_NHCW);
   }
#endif


   /*
    * Compare Ref model network output tensor with Apex inference network's output tensor
    * Beware, Ref model network output tensor should be compact N-C-H-W tensor while Apex network might be
    * N-H-C-W format and does not have to be compact. The comparsion won't care the format and only compare

    * the pixel values at corresponding coordinate.
    */
   if (apexdnnCompare4dTensorDescriptor(Ref_NetOutputTensor,
      Apex_NetOutputTensor) != APEXDNN_STATUS_SUCCESS)
   {
      reval |= 1;
   }

   apexdnnDestroyNet(Ref_Net,  Ref_NetInputTensor);
   apexdnnDestroyNet(Apex_Net, Apex_NetInputTensor);
   apexdnnDestroyTensorDescriptor(TempTensor);
   apexdnnDestroyWorkSpace(Workspace);
   free(ModelBuf);
```

```
   const int64_t cOalMemoryFreeSize_after = OAL_MemorySizeFree();
   printf("%s Before [%d] After[%d], Total[%d]\n", __FILE__, cOalMemoryFreeSize_before,
      cOalMemoryFreeSize_after, OAL_MemorySizeTotal());
   if(cOalMemoryFreeSize_after != cOalMemoryFreeSize_before)
   {
      printf("%s : Memory Leak Detected[%d], Total[%d]\n", __FILE__, cOalMemoryFreeSize_before -
      cOalMemoryFreeSize_after, OAL_MemorySizeTotal());
   }

   if (reval)
   {
      SWT_ARM_LOG_FUNCTION_RVAL_UPDATE(reval, "miss-match ref");
      printf("\t--- Done (***FAIL***)\n");
   }
   else
   {
      printf("\t--- Done (PASS)\n");
   }

   return reval;
}
*
```

Chapter 8

Module Documentation

8.1 UserAPI

Enumerations

- enum apexdnnEltWiseOpType_t { APEXDNN_ELTWISE_OP_UNKNOWN = -1, APEXDNN_ELTWISE_OP_MUL = 0, APEXDNN_ELTWISE_OP_ADD = 1 }

  APEX-DNN supported element-wise operation.
- enum apexdnnLayerType_t { APEXDNN_LAYER_UNKNOWN = -1, APEXDNN_MODULE_TYPE_CONV3X3MPS1 = 0, APEXDNN_MODULE_TYPE_E1E3MPS1 = 1, APEXDNN_MODULE_TYPE_E1E3MP = 2, APEXDNN_MODULE_TYPE_E1E3S1 = 3, APEXDNN_MODULE_TYPE_E1E3 = 4, APEXDNN_MODULE_TYPE_E1AP = 5, APEXDNN_MODULE_TYPE_ELTMULCRED = 6 }

  APEX-DNN supported layer and module types.
- enum apexdnnStatus_t { APEXDNN_STATUS_SUCCESS = 0, APEXDNN_STATUS_BAD_PARAM = 1, APEXDNN_STATUS_ALLOC_FAILED = 2, APEXDNN_STATUS_INTERNAL_ERROR = 3, APEXDNN_STATUS_WORKSPACE_ERROR = 4, APEXDNN_STATUS_HASNOT_VERIFY_GRAPH = 5 }

  APEX-DNN status.
- enum apexdnnTensorDataType_t { APEXDNN_DATA_8BIT = 0, APEXDNN_DATA_16BIT = 1 }

  APEX-DNN supported tensor data type.
- enum apexdnnTensorMemory_t { APEXDNN_TENSOR_MEM_HEAP = 0, APEXDNN_TENSOR_MEM_OAL = 1 }

  APEX-DNN supported memory the tensor will be allocated on.
- enum apexdnnPoolingMode_t { APEXDNN_POOLING_MAX = 0, APEXDNN_POOLING_AVG = 1 }

  APEX-DNN supported pooling mode.
- enum apexdnnShow_Lvl { APEXDNN_SHOW_SIMPLE = 0, APEXDNN_SHOW_VERBOSE = 1, APEXDNN_SHOW_SUPER = 2 }

  APEX-DNN supported show level.

Functions

- apexdnnStatus_t apexdnnCreateWorkSpace (apexdnnWorkSpace **Workspace, ACF_APU_CFG mApuConfig, int mApexID)

  Create APEX-DNN library's workspace.

- apexdnnStatus_t apexdnnDestroyWorkSpace (apexdnnWorkSpace *Workspace)

  Destroy APEX-DNN library's workspace and free up the workspace reserved memory.
- apexdnnStatus_t apexdnnCreate4dTensorDescriptor (apexdnnTensorDescriptor **TensorDesc, apexdnnTensorDataType_t DataType, apexdnnTensorFormat_t Format, apexdnnTensorMemory_t Memory, int dim0, int dim1, int dim2, int dim3, int stride0, int stride1, int stride2, int stride3)

  Create APEX-DNN library's 4-D tensor.
- apexdnnStatus_t apexdnnCreate4dTensorDescriptor (apexdnnTensorDescriptor **TensorDesc, apexdnnTensorDataType_t DataType, apexdnnTensorFormat_t Format, apexdnnTensorMemory_t Memory, int dim0, int dim1, int dim2, int dim3)

  Create APEX-DNN library's 4-D compact tensor.
- apexdnnStatus_t apexdnnCreateVirtual4dTensorDescriptor (apexdnnTensorDescriptor **TensorDesc, apexdnnTensorDataType_t DataType, apexdnnTensorFormat_t Format, apexdnnTensorMemory_t Memory, int dim0, int dim1, int dim2, int dim3)

  Create APEX-DNN library's virtual 4-D compact tensor.
- void * apexdnnRetTensorDataPtr (apexdnnTensorDescriptor *TensorDesc)

  Return tensor's data pointer.
- apexdnnTensorFormat_t apexdnnRetTensorFormat (apexdnnTensorDescriptor *TensorDesc)

  Return tensor's data format.
- apexdnnTensorMemory_t apexdnnRetTensorMemory (apexdnnTensorDescriptor *TensorDesc)

  Return tensor's buffer memory.
- int apexdnnRetTensorDim (apexdnnTensorDescriptor *TensorDesc, int i)

  Return tensor's i-th dimension.
- apexdnnStatus_t apexdnnTransform4dTensorDescriptor (apexdnnTensorDescriptor *XTensorDesc, apexdnnTensorDescriptor *YTensorDesc)

  transform 4-D tensor's data buffer into different format.
- apexdnnStatus_t apexdnnCompare4dTensorDescriptor (apexdnnTensorDescriptor *XTensorDesc, apexdnnTensorDescriptor *YTensorDesc)

  Compre two 4-D tensor's data buffer content.
- apexdnnStatus_t apexdnnRandomize4dTensorDescriptor (apexdnnTensorDescriptor *TensorDesc, int Seed)

  Randomize the tensor.
- void apexdnnTensorDescriptorShow (apexdnnTensorDescriptor *Tensor)

  display tensor descriptor
- apexdnnStatus_t apexdnnDestroyTensorDescriptor (apexdnnTensorDescriptor *TensorDesc)

  destroy tensor descriptor
- apexdnnStatus_t apexdnnCreateConv3x3MPS1Module (apexdnnConv3x3MPS1Module **Module, apexdnnTensorDataType_t DataType, int wConv3x3Dim0, int wConv3x3Dim1, int Conv3x3PadH, int Conv3x3PadW, int Conv3x3StrideH, int Conv3x3StrideW, int MPWindowH, int MPWindowW, int MPPadH, int MPPadW, int MPStrideH, int MPStrideW, int mS1Dim0, int ConvS1StrideH, int COnvS1StrideW)

  Create Conv3x3MPS1 convolution network module.
- apexdnnStatus_t apexdnnCreateE1E3MPS1Module (apexdnnE1E3MPS1Module **Module, apexdnnTensorDataType_t DataType, int wE1Dim0, int wE1Dim1, int E1StrideH, int E1StrideW, int wE3Dim0, int wE3Dim1, int E3PadH, int E3PadW, int E3StrideH, int E3StrideW, int MPWindowH, int MPWindowW, int MPPadH, int MPPadW, int MPStrideH, int MPStrideW, int wS1Dim0, int S1StrideH, int S1StrideW)

  Create E1E3MPS1 convolution network module.
- apexdnnStatus_t apexdnnCreateE1E3MPModule (apexdnnE1E3MPModule **Module, apexdnnTensorDataType_t DataType, int wE1Dim0, int wE1Dim1, int E1StrideH, int E1StrideW, int wE3Dim0, int wE3Dim1, int E3PadH, int E3PadW, int E3StrideH, int E3StrideW, int MPWindowH, int MPWindowW, int MPPadH, int MPPadW, int MPStrideH, int MPStrideW)

  Create E1E3MP convolution network module.

- apexdnnStatus_t apexdnnCreateE1E3S1Module (apexdnnE1E3S1Module **Module, apexdnnTensorDataType_t DataType, int wE1Dim0, int wE1Dim1, int E1StrideH, int E1StrideW, int wE3Dim0, int wE3Dim1, int E3PadH, int E3PadW, int E3StrideH, int E3StrideW, int wS1Dim0, int S1StrideH, int S1StrideW)

  Create E1E3S1 convolution network module.
- apexdnnStatus_t apexdnnCreateE1E3Module (apexdnnE1E3Module **Module, apexdnnTensorDataType_t DataType, int wE1Dim0, int wE1Dim1, int E1StrideH, int E1StrideW, int wE3Dim0, int wE3Dim1, int E3PadH, int E3PadW, int E3StrideH, int E3StrideW)

  Create E1E3 convolution network module.
- apexdnnStatus_t apexdnnCreateEmptyNet (apexdnnNet **Net, bool BatchProc=0)

  Create convolution neural network.
- apexdnnStatus_t apexdnnDestroyNet (apexdnnNet *Net, apexdnnTensorDescriptor *NetInputTensorDesc)

  Destroy convolution neural network.
- int apexdnnRetNetNumofLayers (apexdnnNet *Net)

  Get total number of layers in the network.
- apexdnnTensorDescriptor * apexdnnRetNetLayerOutputTensorDesc (apexdnnNet *Net, int i)

  Get each layer's output tensor.
- apexdnnLayerType_t apexdnnRetNetLayerType (apexdnnNet *Net, int i)

  Get each layer's type.
- apexdnnStatus_t apexdnnNetAppendLayer (apexdnnNet *Net, void *Layer)

  Add layer or module into network.
- apexdnnStatus_t apexdnnNetFillModel (apexdnnNet *Net, int8_t *Model)

  Fill network model (weights, bias, quantization parameters) into network.
- apexdnnStatus_t apexdnnNetVerifyGraphCpu (apexdnnNet *Net, apexdnnTensorDescriptor *NetInputTensorDesc, apexdnnTensorDescriptor **NetOutputTensorDesc)

  Verify network.
- apexdnnStatus_t apexdnnNetForwardCpu (apexdnnWorkSpace *Workspace, apexdnnNet *Net, apexdnnTensorDescriptor *NetInputTensorDesc)

  Forward network.
- apexdnnStatus_t apexdnnNetVerifyGraphApex (apexdnnWorkSpace *Workspace, apexdnnNet *Net, apexdnnTensorDescriptor *NetInputTensorDesc, apexdnnTensorDescriptor **NetOutputTensorDesc)

  Verify network.
- apexdnnStatus_t apexdnnNetForwardApex (apexdnnWorkSpace *Workspace, apexdnnNet *Net, apexdnnTensorDescriptor *NetInputTensorDesc, int Profiling=0)

  Forward network.
- void apexdnnNetShow (apexdnnNet *Net, apexdnnShow_Lvl Lvl=APEXDNN_SHOW_SIMPLE)

  Display Netowork.

8.1.1 Detailed Description

This is the group of enum, structure and functions needs to be exposed to APEX-DNN library user

8.1.2 Enumeration Type Documentation

8.1.2.1 enum apexdnnEltWiseOpType_t

APEX-DNN supported element-wise operation.

apexdnnEltWiseOpType_t is a enumerated type used to declares all APEX-DNN library supported element-wise operation.

Enumerator

APEXDNN_ELTWISE_OP_UNKNOWN Undefined type

APEXDNN_ELTWISE_OP_MUL Element-wise multiplication

APEXDNN_ELTWISE_OP_ADD Element-wise addition

8.1.2.2 enum apexdnnLayerType_t

APEX-DNN supported layer and module types.

apexdnnLayerType_t is a enumerated type used to declares all APEX-DNN library supported layers and modules. We use term "LAYER" if one only one operation, such as convolution, max pooling or ReLu, is executed; we use term "MODULE" if multiple operations are executed.

Enumerator

APEXDNN_LAYER_UNKNOWN Undefined type

APEXDNN_MODULE_TYPE_CONV3X3MPS1 Module includes 3x3 convolution filter, followed by max pooling filter then squeeze 1x1 convolution filter

APEXDNN_MODULE_TYPE_E1E3MPS1 Module includes expand 1x1 convolution filter and expand 3x3 convolution filter followed by max pooling filter then squeeze 1x1 convolution filter

APEXDNN_MODULE_TYPE_E1E3MP Module includes expand 1x1 convolution filter and expand 3x3 convolution filter followed by max pooling filter

APEXDNN_MODULE_TYPE_E1E3S1 Module includes expand 1x1 convolution filter and expand 3x3 convolution filter followed by squeeze 1x1 convolution filter

APEXDNN_MODULE_TYPE_E1E3 Module includes expand 1x1 convolution filter and expand 3x3 convolution filter

APEXDNN_MODULE_TYPE_E1AP Module includes expand 1x1 convolution filter and expand 3x3 convolution filter

APEXDNN_MODULE_TYPE_ELTMULCRED Layer to do element wise operation. Only support multiplication for now.

8.1.2.3 enum apexdnnPoolingMode_t

APEX-DNN supported pooling mode.

apexdnnPoolingMode_t is a enumberated type used to declare differnet pooling mode.

Enumerator

APEXDNN_POOLING_MAX max pooling which is the only supported pooling mode for now

APEXDNN_POOLING_AVG average pooling

8.1.2.4 enum apexdnnShow_Lvl

APEX-DNN supported show level.

apexdnnShow_Lvl is a enumberated type used to declare different show level when user call Show routine on network, tensor or each layer/module for debugging purpose.

Enumerator

APEXDNN_SHOW_SIMPLE Basic level to display most critical information only

APEXDNN_SHOW_VERBOSE Verbose level

APEXDNN_SHOW_SUPER Unused yet

8.1.2.5 enum apexdnnStatus_t

APEX-DNN status.

apexdnnStatus_t is a enumerated type used to declare function returned status.

Enumerator

APEXDNN_STATUS_SUCCESS The operation completed successfully

APEXDNN_STATUS_BAD_PARAM An incorrect value or parameter was passed to the function

APEXDNN_STATUS_ALLOC_FAILED Memory allocation failed

APEXDNN_STATUS_INTERNAL_ERROR An internal apexdnn executation failed

APEXDNN_STATUS_WORKSPACE_ERROR Workspace hasn't been allocated yet

APEXDNN_STATUS_HASNOT_VERIFY_GRAPH Network graph hasn't been verified yet

8.1.2.6 enum apexdnnTensorDataType_t

APEX-DNN supported tensor data type.

apexdnnTensorDataType_t is a enumerated type used to declare supported data types.

Enumerator

APEXDNN_DATA_8BIT Fixed point 8-bit, the only one supported for now

APEXDNN_DATA_16BIT Fixed point 16-bit, not supported yet

8.1.2.7 enum apexdnnTensorMemory_t

APEX-DNN supported memory the tensor will be allocated on.

apexdnnTensorMemory_t is a enumberated type used to declare on which memory partation the tensor will be allocated.

Enumerator

APEXDNN_TENSOR_MEM_HEAP regular heap memory

APEXDNN_TENSOR_MEM_OAL OAL memory

8.1.3 Function Documentation

8.1.3.1 apexdnnStatus_t apexdnnCompare4dTensorDescriptor (apexdnnTensorDescriptor * *XTensorDesc,* apexdnnTensorDescriptor * *YTensorDesc*)

Compre two 4-D tensor's data buffer content.

This is an utilit to compare two tensors buffer content. The comparison won't care the tensors' format and only compare the pixel values at corresponding coordinate. Only support the tensors of N-C-H-W and N-H-C-W format.

8.1.3.2 apexdnnStatus_t apexdnnCreate4dTensorDescriptor (apexdnnTensorDescriptor ∗∗ *TensorDesc,* apexdnnTensorDataType_t *DataType,* apexdnnTensorFormat_t *Format,* apexdnnTensorMemory_t *Memory,* int *dim0,* int *dim1,* int *dim2,* int *dim3,* int *stride0,* int *stride1,* int *stride2,* int *stride3*)

Create APEX-DNN library's 4-D tensor.

This is an interface for creating the 4-D tensor descriptor by allocating the memory needed to hold its data structure. The tensor can have padding on each dimension when the stride is greather than product of the dimension and the stride of the next dimension (stride[n] $>$ stride[n+1] $*$ dim[n+1]).

Parameters

TensorDesc	Pointer point to the tensor's handle
DataType	Tensor data type, only 8-bit is supported for now
Format	Tensor layout format
Memory	Tensor memory
dim0	Size of outer most dimension
dim1	Size of second dimension
dim2	Size of third dimension
dim3	Size of inner most dimension
stride0	Outer most dimension's stride
stride1	Second dimension's stride
stride2	Third dimension's stride
stride3	Inner most dimension's stride

8.1.3.3 apexdnnStatus_t apexdnnCreate4dTensorDescriptor (apexdnnTensorDescriptor ∗∗ *TensorDesc,* apexdnnTensorDataType_t *DataType,* apexdnnTensorFormat_t *Format,* apexdnnTensorMemory_t *Memory,* int *dim0,* int *dim1,* int *dim2,* int *dim3*)

Create APEX-DNN library's 4-D compact tensor.

This is an interface for creating the 4-D tensor descriptor by allocating the memory needed to hold its data structure. This will create a compact tensor, ie. the stride of one dimension equal to the product of the dimension and the stride of the next dimension (stride[n] = stride[n+1] $*$ dim[n+1]).

Parameters

TensorDesc	Pointer point to the tensor's handle
DataType	Tensor data type, only 8-bit is supported for now
Format	Tensor layout format
Memory	Tensor memory
dim0	Size of outer most dimension
dim1	Size of second dimension
dim2	Size of third dimension
dim3	Size of inner most dimension

8.1.3.4 apexdnnStatus_t apexdnnCreateConv3x3MPS1Module (apexdnnConv3x3MPS1Module ∗∗ *Module,* apexdnnTensorDataType_t *DataType,* int *wConv3x3Dim0,* int *wConv3x3Dim1,* int *Conv3x3PadH,* int *Conv3x3PadW,* int *Conv3x3StrideH,* int *Conv3x3StrideW,* int *MPWindowH,* int *MPWindowW,* int *MPPadH,* int *MPPadW,* int *MPStrideH,* int *MPStrideW,* int *mS1Dim0,* int *ConvS1StrideH,* int *COnvS1StrideW*)

Create Conv3x3MPS1 convolution network module.

This is an interface to create a convolution network module which includes 3x3 convolution filter, followed by max pooling

filter then squeeze 1x1 convolution filter.

Parameters

Module	Pointer which points to the module's handle
DataType	Data type, only support 8-bit for now
wConv3x3Dim0	3x3 convolution filter's number of output channels
wConv3x3Dim1	3x3 convolution filter's number of input channels
Conv3x3PadH	3x3 convolution filter's vertical padding
Conv3x3PadW	3x3 convolution filter's horizontal padding
Conv3x3StrideH	3x3 convolution filter's vertical stride
Conv3x3StrideW	3x3 convolution filter's horizontal stride
MPWindowH	Maxpooling filter's vertical window size
MPWindowW	Maxpooling filter's horizontal window size
MPPadH	Maxpooling filter's vertical padding
MPPadW	Maxpooling filter's horizontal padding
MPStrideH	Maxpooling filter's vertical stride
MPStrideW	Maxpooling filter's horizontal stride
mS1Dim0	1x1 convolution filter's number of output channels
ConvS1StrideH	1x1 convolution filter's vertical stride
COnvS1StrideW	1x1 convolution filter's horizontal stride

8.1.3.5 apexdnnStatus_t apexdnnCreateE1E3Module (apexdnnE1E3Module ∗∗ *Module,* apexdnnTensorDataType_t *DataType,* int *wE1Dim0,* int *wE1Dim1,* int *E1StrideH,* int *E1StrideW,* int *wE3Dim0,* int *wE3Dim1,* int *E3PadH,* int *E3PadW,* int *E3StrideH,* int *E3StrideW*)

Create E1E3 convolution network module.

This is an interface to create a convolution network module which includes expand 1x1 convolution filter and expand 3x3 convolution filter.

Parameters

Module	Pointer which points to the module's handle
DataType	Data type, only support 8-bit for now
wE1Dim0	1x1 convolution filter's number of output channels
wE1Dim1	1x1 convolution filter's number of input channels
E1StrideH	1x1 convolution filter's vertical stride
E1StrideW	1x1 convolution filter's horizontal stride
wE3Dim0	3x3 convolution filter's number of output channels
wE3Dim1	3x3 convolution filter's number of input channels
E3PadH	3x3 convolution filter's vertical padding
E3PadW	3x3 convolution filter's horizontal padding
E3StrideH	3x3 convolution filter's vertical stride
E3StrideW	3x3 convolution filter's horizontal stride

8.1.3.6 apexdnnStatus_t apexdnnCreateE1E3MPModule (apexdnnE1E3MPModule ∗∗ *Module,* apexdnnTensorDataType_t *DataType,* int *wE1Dim0,* int *wE1Dim1,* int *E1StrideH,* int *E1StrideW,* int *wE3Dim0,* int *wE3Dim1,* int *E3PadH,* int *E3PadW,* int *E3StrideH,* int *E3StrideW,* int *MPWindowH,* int *MPWindowW,* int *MPPadH,* int *MPPadW,* int *MPStrideH,* int *MPStrideW*)

Create E1E3MP convolution network module.

This is an interface to create a convolution network module which includes expand 1x1 convolution filter and expand 3x3 convolution filter followed by max pooling filter.

Parameters

Module	Pointer which points to module's handle
DataType	Data type, only support 8-bit for now
wE1Dim0	1x1 convolution filter's number of output channels
wE1Dim1	1x1 convolution filter's number of input channels
E1StrideH	1x1 convolution filter's vertical stride
E1StrideW	1x1 convolution filter's horizontal stride
wE3Dim0	3x3 convolution filter's number of output channels
wE3Dim1	3x3 convolution filter's number of input channels
E3PadH	3x3 convolution filter's vertical padding
E3PadW	3x3 convolution filter's horizontal padding
E3StrideH	3x3 convolution filter's vertical stride
E3StrideW	3x3 convolution filter's horizontal stride
MPWindowH	Maxpooling filter's vertical window size
MPWindowW	3x3 convolution filter's horizontal stride
MPPadH	Maxpooling filter's vertical padding
MPPadW	Maxpooling filter's horizontal padding
MPStrideH	Maxpooling filter's vertical stride
MPStrideW	Maxpooling filter's horizontal stride

8.1.3.7 apexdnnStatus_t apexdnnCreateE1E3MPS1Module (apexdnnE1E3MPS1Module ∗∗ *Module,* apexdnnTensorDataType_t *DataType,* int *wE1Dim0,* int *wE1Dim1,* int *E1StrideH,* int *E1StrideW,* int *wE3Dim0,* int *wE3Dim1,* int *E3PadH,* int *E3PadW,* int *E3StrideH,* int *E3StrideW,* int *MPWindowH,* int *MPWindowW,* int *MPPadH,* int *MPPadW,* int *MPStrideH,* int *MPStrideW,* int *wS1Dim0,* int *S1StrideH,* int *S1StrideW*)

Create E1E3MPS1 convolution network module.

This is an interface to create a convolution network module which includes expand 1x1 convolution filter and expand 3x3 convolution filter followed by max pooling filter then squeeze 1x1 convolution filter.

Parameters

Module	Pointer which points to module's handle
DataType	Data type, only support 8-bit for now
wE1Dim0	1x1 convolution filter's number of output channels
wE1Dim1	1x1 convolution filter's number of input channels
E1StrideH	1x1 convolution filter's vertical stride
E1StrideW	1x1 convolution filter's horizontal stride
wE3Dim0	3x3 convolution filter's number of output channels
wE3Dim1	3x3 convolution filter's number of input channels
E3PadH	3x3 convolution filter's vertical padding
E3PadW	3x3 convolution filter's horizontal padding
E3StrideH	3x3 convolution filter's vertical stride
E3StrideW	3x3 convolution filter's horizontal stride
MPWindowH	Maxpooling filter's vertical window size
MPWindowW	3x3 convolution filter's horizontal stride

MPPadH	Maxpooling filter's vertical padding
MPPadW	Maxpooling filter's horizontal padding
MPStrideH	Maxpooling filter's vertical stride
MPStrideW	Maxpooling filter's horizontal stride
wS1Dim0	1x1 convolution filter's number of output channels
S1StrideH	1x1 convolution filter's vertical stride
S1StrideW	1x1 convolution filter's horizontal stride

8.1.3.8 apexdnnStatus_t apexdnnCreateE1E3S1Module (apexdnnE1E3S1Module ** *Module,* apexdnnTensorDataType_t *DataType,* int *wE1Dim0,* int *wE1Dim1,* int *E1StrideH,* int *E1StrideW,* int *wE3Dim0,* int *wE3Dim1,* int *E3PadH,* int *E3PadW,* int *E3StrideH,* int *E3StrideW,* int *wS1Dim0,* int *S1StrideH,* int *S1StrideW*)

Create E1E3S1 convolution network module.

This is an interface to create a convolution network module which includes expand 1x1 convolution filter and expand 3x3 convolution filter followed by squeeze 1x1 convolution filter.

Parameters

Module	Pointer which points to the module's handle
DataType	Data type, only support 8-bit for now
wE1Dim0	1x1 convolution filter's number of output channels
wE1Dim1	1x1 convolution filter's number of input channels
E1StrideH	1x1 convolution filter's vertical stride
E1StrideW	1x1 convolution filter's horizontal stride
wE3Dim0	3x3 convolution filter's number of output channels
wE3Dim1	3x3 convolution filter's number of input channels
E3PadH	3x3 convolution filter's vertical padding
E3PadW	3x3 convolution filter's horizontal padding
E3StrideH	3x3 convolution filter's vertical stride
E3StrideW	3x3 convolution filter's horizontal stride
wS1Dim0	1x1 convolution filter's number of output channels
S1StrideH	1x1 convolution filter's vertical stride
S1StrideW	1x1 convolution filter's horizontal stride

8.1.3.9 apexdnnStatus_t apexdnnCreateEmptyNet (apexdnnNet ** *Net,* bool *BatchProc* = 0)

Create convolution neural network.

This is an interface for creating an empty convolution neural network by allocating the memory needed to hold its data structure.

Parameters

Net	Pointer which points to the network's handle
BatchProc	Batch processing enablement. 0: Disabled. 1: Enabled

8.1.3.10 apexdnnStatus_t apexdnnCreateVirtual4dTensorDescriptor (apexdnnTensorDescriptor ** *TensorDesc,* apexdnnTensorDataType_t *DataType,* apexdnnTensorFormat_t *Format,* apexdnnTensorMemory_t *Memory,* int *dim0,* int *dim1,* int *dim2,* int *dim3*)

Create APEX-DNN library's virtual 4-D compact tensor.

This is an interface for creating the virtual 4-D tensor descriptor but not really allocate the memory This will create a compact virtual tensor, ie. the stride of one dimension equal to the product of the dimension and the stride of the next dimension (stride[n] = stride[n+1] ∗ dim[n+1]).

Parameters

TensorDesc	Pointer point to the tensor's handle
DataType	Tensor data type, only 8-bit is supported for now
Format	Tensor layout format
Memory	Tensor memory
dim0	Size of outer most dimension
dim1	Size of second dimension
dim2	Size of third dimension
dim3	Size of inner most dimension

8.1.3.11 apexdnnStatus_t apexdnnCreateWorkSpace (apexdnnWorkSpace ∗∗ *Workspace,* ACF_APU_CFG *mApuConfig,* int *mApexID*)

Create APEX-DNN library's workspace.

This is an interface for creating the APEX-DNN library's workspace by allocating the memory needed to hold its data structure.

Parameters

Workspace	Workspace handle
mApuConfig	Apu configuration, only support ACF_APU_CFG__APU_0_CU_0_63_SMEM_0_3 for now
mApexID	APEX ID

8.1.3.12 apexdnnStatus_t apexdnnDestroyNet (apexdnnNet ∗ *Net,* apexdnnTensorDescriptor ∗ *NetInputTensorDesc*)

Destroy convolution neural network.

This is an interface for destroying an convolution neural network. All the layers and related tensors will be destroied and allocated memory will be freed.

Parameters

Net	Network to be destroied
NetInputTensor-Desc	Network's input tensor to be destroied

8.1.3.13 apexdnnStatus_t apexdnnDestroyTensorDescriptor (apexdnnTensorDescriptor ∗ *TensorDesc*)

destroy tensor descriptor

This is an interface to destroy tensor descriptor. Tensor's memory will be freed.

Parameters

TensorDesc	Tensor descriptor needs to be destroyed

8.1.3.14 apexdnnStatus_t apexdnnDestroyWorkSpace (apexdnnWorkSpace * *Workspace*)

Destroy APEX-DNN library's workspace and free up the workspace reserved memory.

This is an interface for destroying the APEX-DNN library's workspace and free up the reserved memory.

Parameters

Workspace	Work space

8.1.3.15 apexdnnStatus_t apexdnnNetAppendLayer (apexdnnNet * *Net,* void * *Layer*)

Add layer or module into network.

This is an interface to add the layer into network.

Parameters

Net	Network
Layer	Created layer needs to add into network

8.1.3.16 apexdnnStatus_t apexdnnNetFillModel (apexdnnNet * *Net,* int8_t * *Model*)

Fill network model (weights, bias, quantization parameters) into network.

This is an interface to fill in network model, including each filter's weight, bias and quantization parameters.

Parameters

Net	Network
Model	Model buffer which hold read in fixed point model file produced by offline tool

8.1.3.17 apexdnnStatus_t apexdnnNetForwardApex (apexdnnWorkSpace * *Workspace,* apexdnnNet * *Net,* apexdnnTensorDescriptor * *NetInputTensorDesc,* int *Profiling* = 0)

Forward network.

This is an interface to forward the network on APEX.

Parameters

Workspace	Work space handle
Net	Network
NetInputTensor-Desc	Network input tesnor
Profiling	Enable ACF profiling. Profiling information will be printed onto console. 0: Disabled. 1: Enabled

8.1.3.18 apexdnnStatus_t apexdnnNetForwardCpu (apexdnnWorkSpace * *Workspace,* apexdnnNet * *Net,* apexdnnTensorDescriptor * *NetInputTensorDesc*)

Forward network.

This is an interface to forward the network on host CPU. This serve as reference model, i.e. very straight forward implementation without any level optimization.

Parameters

Workspace	Work space handle
Net	Network
NetInputTensor-Desc	Network input tesnor

8.1.3.19 void apexdnnNetShow (apexdnnNet * *Net,* apexdnnShow_Lvl *Lvl* = APEXDNN_SHOW_SIMPLE)

Display Netowork.

This is a debug utility function to show the network's configuration.

Parameters

Net	Network
Lvl	Show level

8.1.3.20 apexdnnStatus_t apexdnnNetVerifyGraphApex (apexdnnWorkSpace * *Workspace,* apexdnnNet * *Net,* apexdnnTensorDescriptor * *NetInputTensorDesc,* apexdnnTensorDescriptor ** *NetOutputTensorDesc*)

Verify network.

This is an interface to verify the network. It won't really perform forward calculation instead it will go through the network, check the consistency and allocate intermediate tensor for each layer or module's output based on input tensor's resolution. This only needs to execute once if no input resolution change. The output tensor if provided will be connect to last layer. Otherwise it will be internally allocated. Thie is for host CPU forwarding, i.e. assume the following forward calculation call will be "apexdnnNetForwardApex".

Parameters

Workspace	Work space handle
Net	Network
NetInputTensor-Desc	Network input tesnor
NetOutput-TensorDesc	Network output tesnor

8.1.3.21 apexdnnStatus_t apexdnnNetVerifyGraphCpu (apexdnnNet * *Net,* apexdnnTensorDescriptor * *NetInputTensorDesc,* apexdnnTensorDescriptor ** *NetOutputTensorDesc*)

Verify network.

This is an interface to verify the network. It won't really perform forward calculation instead it will go through the network, check the consistency and allocate intermediate tensor for each layer or module's output based on input tensor's resolution. This only needs to execute once if no input resolution change. The output tensor if provided will be connect to last layer. Otherwise it will be internally allocated. Thie is for host CPU forwarding, i.e. assume the following forward calculation call will be "apexdnnNetForwardCpu".

Parameters

Net	Network
NetInputTensor-Desc	Network input tesnor
NetOutput-TensorDesc	Network output tesnor

8.1.3.22 apexdnnStatus_t apexdnnRandomize4dTensorDescriptor (apexdnnTensorDescriptor * *TensorDesc,* int *Seed*)

Randomize the tensor.

This is an utilit to fill in tensor with random value. Mainly for testing / debugging purpose.

8.1.3.23 apexdnnTensorDescriptor* apexdnnRetNetLayerOutputTensorDesc (apexdnnNet * *Net,* int *i*)

Get each layer's output tensor.

This is an interface for retrieving the output tensor of each network's layer.

Parameters

Net	Network
i	Layer index

8.1.3.24 apexdnnLayerType_t apexdnnRetNetLayerType (apexdnnNet * *Net,* int *i*)

Get each layer's type.

This is an interface for retrieving the type of each network's layer.

Parameters

Net	Network
i	Layer index

8.1.3.25 int apexdnnRetNetNumofLayers (apexdnnNet * *Net*)

Get total number of layers in the network.

This is an interface for retrieving the total number of layers in the network.

Parameters

Net	Network

8.1.3.26 void* apexdnnRetTensorDataPtr (apexdnnTensorDescriptor * *TensorDesc*)

Return tensor's data pointer.

This is an interface for returning tensor's data pointer.

Parameters

TensorDesc	Tensor descriptor

8.1.3.27 int apexdnnRetTensorDim (apexdnnTensorDescriptor * *TensorDesc,* int *i*)

Return tensor's i-th dimension.

This is an interface for returning tensor's i-th dimension.

Parameters

TensorDesc	Tensor descriptor
i	Dimension index

8.1.3.28 apexdnnTensorFormat_t apexdnnRetTensorFormat (apexdnnTensorDescriptor * *TensorDesc*)

Return tensor's data format.

This is an interface for returning tensor's data format.

Parameters

TensorDesc	Tensor descriptor

8.1.3.29 apexdnnTensorMemory_t apexdnnRetTensorMemory (apexdnnTensorDescriptor * *TensorDesc*)

Return tensor's buffer memory.

This is an interface for returning tensor's buffer memory.

Parameters

TensorDesc	Tensor descriptor

8.1.3.30 void apexdnnTensorDescriptorShow (apexdnnTensorDescriptor * *Tensor*)

display tensor descriptor

This is an interface to display tensor descriptor for debugging purpose

Parameters

Tensor	Tensor descriptor needs to be displayed

8.1.3.31 apexdnnStatus_t apexdnnTransform4dTensorDescriptor (apexdnnTensorDescriptor * *XTensorDesc,* apexdnnTensorDescriptor * *YTensorDesc*)

transform 4-D tensor's data buffer into different format.

This is an utilit to transform tensor from one format to another. Two tensors have to have separated buffer allocated. Don't support inplace transformation for now. This is unoptimized routine yet for functionality only. Only support following transformation: From N-C-H-W to N-H-C-w From N-H-C-W to N-C-H-W From N-C-H-W to N-H-W-C

Index