Optimize Resnet with efficient GPU kernels

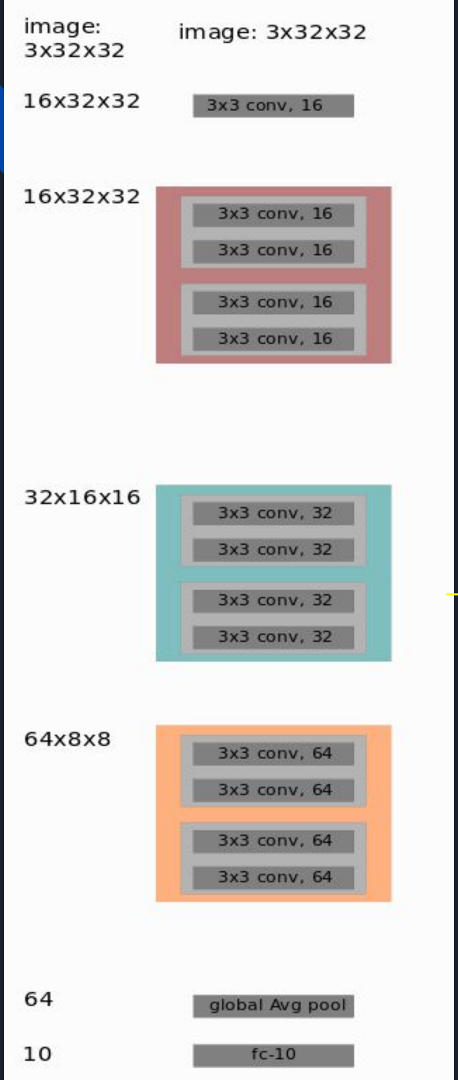
-Project Final Presentation.

Feng Li, Christopher Goebel and Yuankun Fu

Outline

- Framework and Network introduction. (Feng)
- Optimized GPU convolution layer. (Chris)
- Performance and Comparison with CUDNN. (Yuankun)

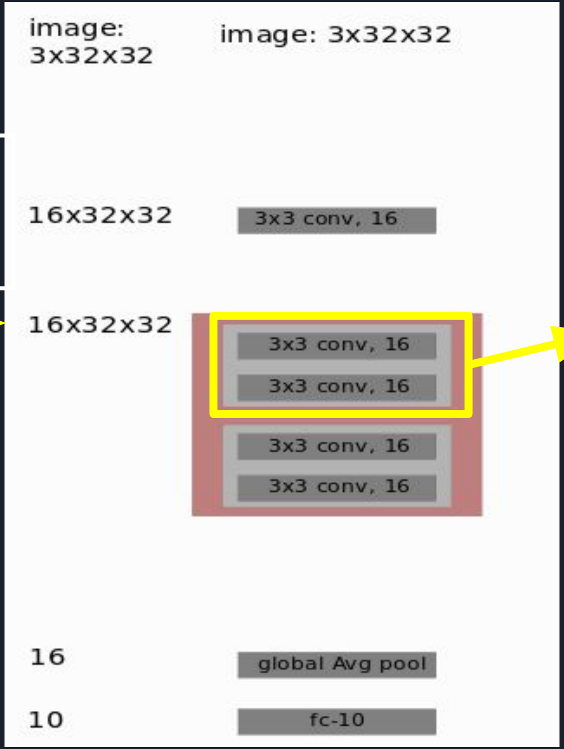
I. Resnet Framework Introduction



Our simplified Resnet

Reduce the stages from the cifar10 experiment in the ResNet paper

Simplified version



Residual block

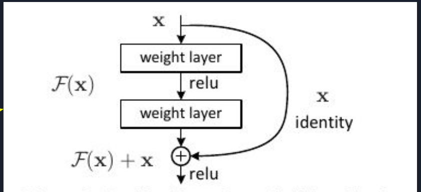
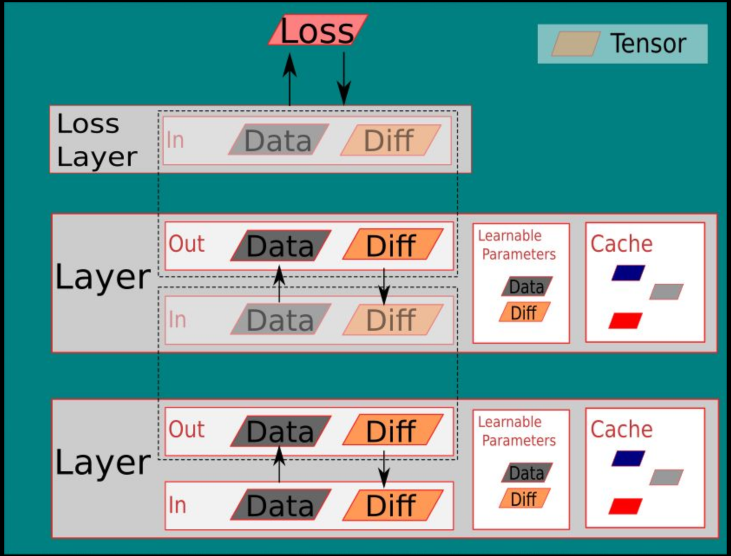

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2. Residual learning: a building block.

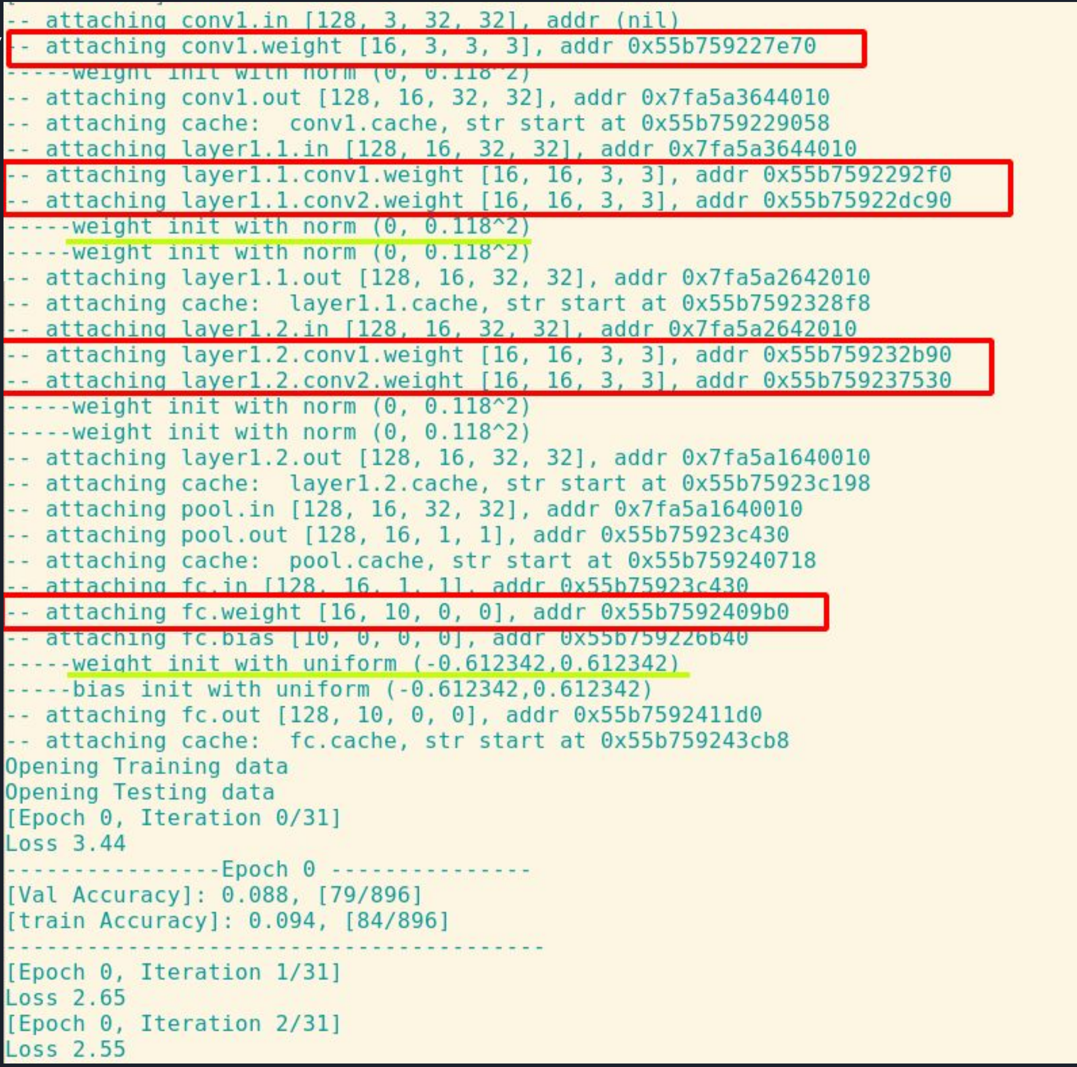
Memory management

- The picture shows memory management of current framework design.
  - Resnet takes images as input, the network is learned so that it can minimize the loss of label predictions.
- **Tensors** data types are used in several places in the network(GPU tensor/cpu tensor.)
  - Layer input/output.
  - Learnable params(e.g. weight/bias in conv/fc layer)
  - Caches. all needed intermediate information for backward, can be a list of tensors.



Example: Prepare Memory

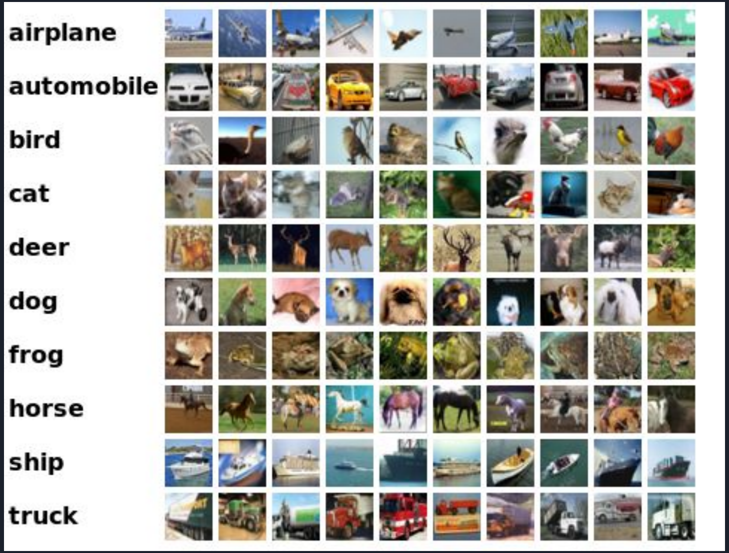
1. Figure in the right show how memory is populated when we start the training.
   - Code is in /tests/test-net-resnet-cifar.
2. Note:
   - Batch size 128.
   - Reuse of layout and Lay input.
3. To train deep network, we also need to initialize weight layer carefully:
   - Convolution layer uses Kai-ming initialization.
   - Fully layer uses uniform distribution.
   - Code at utils/weight_init.cpp.



```
-- attaching conv1.in [128, 3, 32, 32], addr (nil)
-- attaching conv1.weight [16, 3, 3, 3], addr 0x55b759227e70
-----weight init with norm (0, 0.118^2)
-- attaching conv1.out [128, 16, 32, 32], addr 0x7fa5a3644010
-- attaching cache:  conv1.cache, str start at 0x55b759229058
-- attaching layer1.1.in [128, 16, 32, 32], addr 0x7fa5a3644010
-- attaching layer1.1.conv1.weight [16, 16, 3, 3], addr 0x55b7592292f0
-- attaching layer1.1.conv2.weight [16, 16, 3, 3], addr 0x55b75922dc90
-----weight init with norm (0, 0.118^2)
-----weight init with norm (0, 0.118^2)
-- attaching layer1.1.out [128, 16, 32, 32], addr 0x7fa5a2642010
-- attaching cache:  layer1.1.cache, str start at 0x55b7592328f8
-- attaching layer1.2.in [128, 16, 32, 32], addr 0x7fa5a2642010
-- attaching layer1.2.conv1.weight [16, 16, 3, 3], addr 0x55b759232b90
-- attaching layer1.2.conv2.weight [16, 16, 3, 3], addr 0x55b759237530
-----weight init with norm (0, 0.118^2)
-----weight init with norm (0, 0.118^2)
-- attaching layer1.2.out [128, 16, 32, 32], addr 0x7fa5a1640010
-- attaching cache:  layer1.2.cache, str start at 0x55b75923c198
-- attaching pool.in [128, 16, 32, 32], addr 0x7fa5a1640010
-- attaching pool.out [128, 16, 1, 1], addr 0x55b75923c430
-- attaching cache:  pool.cache, str start at 0x55b759240718
-- attaching fc.in [128, 16, 1, 1], addr 0x55b75923c430
-- attaching fc.weight [16, 10, 0, 0], addr 0x55b7592409b0
-- attaching fc.bias [10, 0, 0, 0], addr 0x55b759226b40
-----weight init with uniform (-0.612342,0.612342)
-----bias init with uniform (-0.612342,0.612342)
-- attaching fc.out [128, 10, 0, 0], addr 0x55b7592411d0
-- attaching cache:  fc.cache, str start at 0x55b759243cb8
Opening Training data
Opening Testing data
[Epoch 0, Iteration 0/31]
Loss 3.44
----------------Epoch 0 --------------
[Val Accuracy]: 0.088, [79/896]
[train Accuracy]: 0.094, [84/896]
----------------------------------------
[Epoch 0, Iteration 1/31]
Loss 2.65
[Epoch 0, Iteration 2/31]
Loss 2.55
```

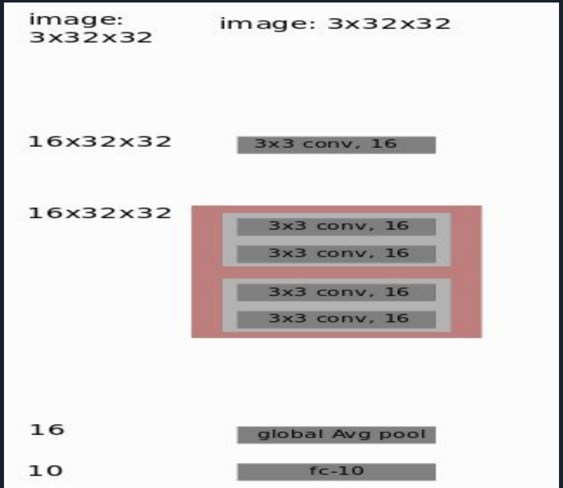
Data preparation

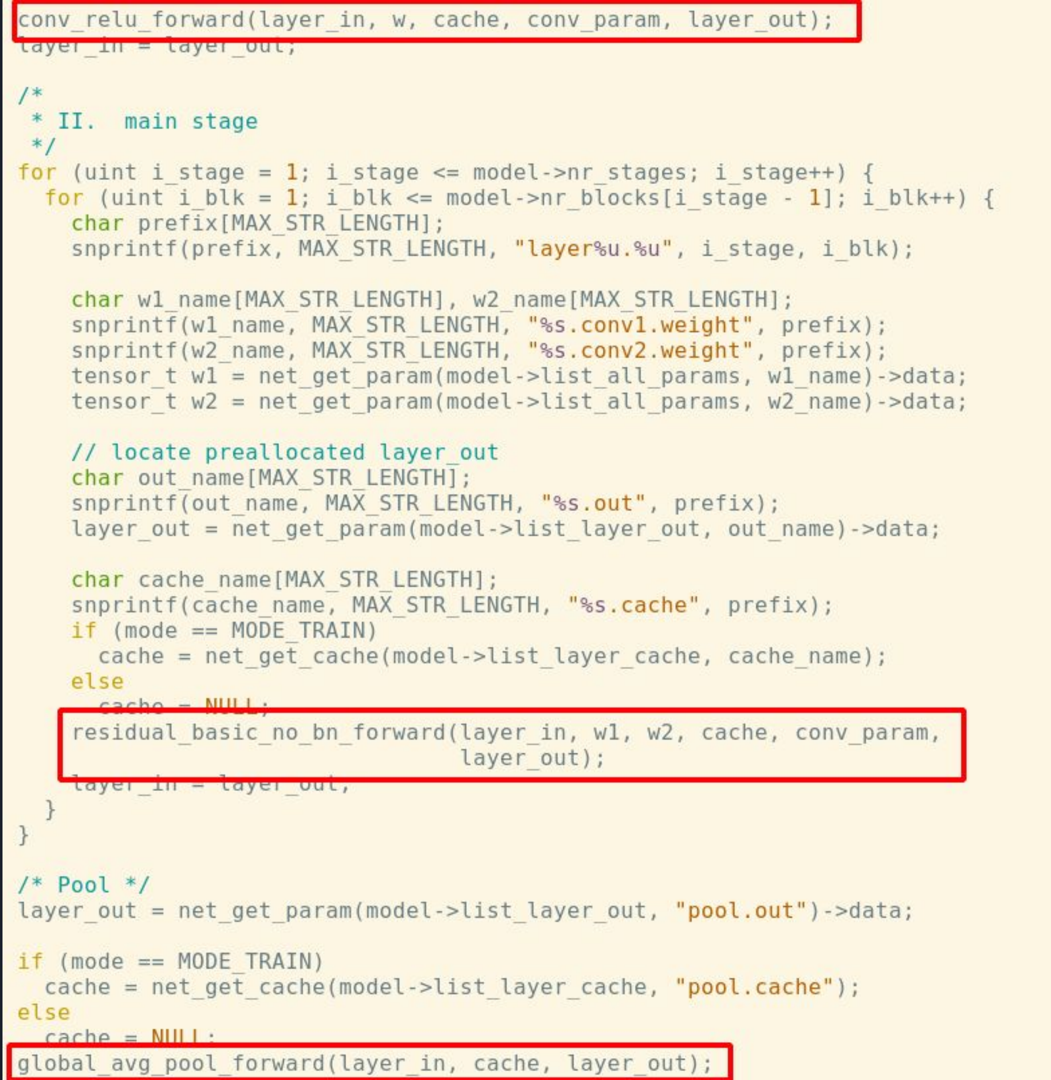
- Cifar10 data loader:
  - Original Data set: 50 000 training img and 10000 testing images (rgb value 0~255).
  - 10 classes.
- Create training set and validation, and feed in batches
  - Subtract channel mean to make each channel fit N(0,1) .
  - Originalt train-img -> training set and validation set.
  - Code is in utils/data_cifar.cpp.



Forward pass

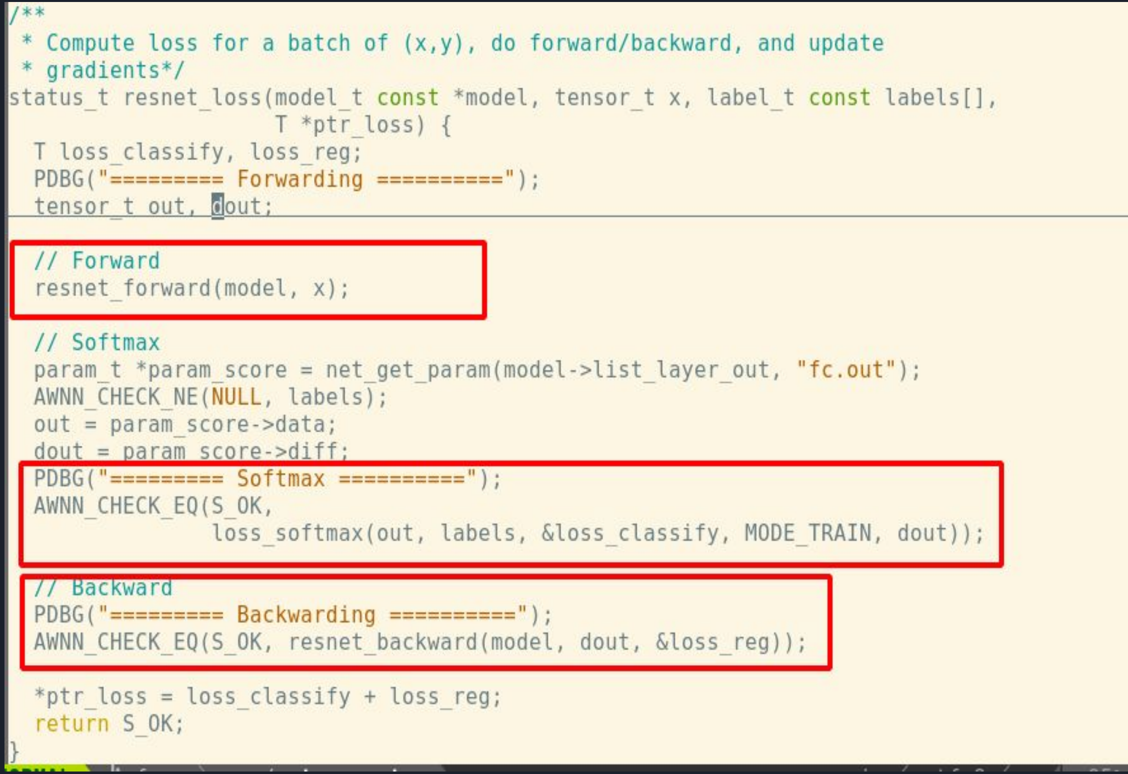
- Forward/Backward
  - Prepare stage-> residual stages-> global_pool + fc
  - src/net_resnet.c
  - Residual_xx_forward function will call convolution layers forward.
- Backward is in similar structure.




```
conv_relu_forward(layer_in, w, cache, conv_param, layer_out);
layer_in = layer_out;

/*
 * II.  main stage
 */
for (uint i_stage = 1; i_stage <= model->nr_stages; i_stage++) {
  for (uint i_blk = 1; i_blk <= model->nr_blocks[i_stage - 1]; i_blk++) {
    char prefix[MAX_STR_LENGTH];
    snprintf(prefix, MAX_STR_LENGTH, "layer%u.%u", i_stage, i_blk);

    char w1_name[MAX_STR_LENGTH], w2_name[MAX_STR_LENGTH];
    snprintf(w1_name, MAX_STR_LENGTH, "%s.conv1.weight", prefix);
    snprintf(w2_name, MAX_STR_LENGTH, "%s.conv2.weight", prefix);
    tensor_t w1 = net_get_param(model->list_all_params, w1_name)->data;
    tensor_t w2 = net_get_param(model->list_all_params, w2_name)->data;

    // locate preallocated layer_out
    char out_name[MAX_STR_LENGTH];
    snprintf(out_name, MAX_STR_LENGTH, "%s.out", prefix);
    layer_out = net_get_param(model->list_layer_out, out_name)->data;

    char cache_name[MAX_STR_LENGTH];
    snprintf(cache_name, MAX_STR_LENGTH, "%s.cache", prefix);
    if (mode == MODE_TRAIN)
      cache = net_get_cache(model->list_layer_cache, cache_name);
    else
      cache = NULL;
    residual_basic_no_bn_forward(layer_in, w1, w2, cache, conv_param,
                                 layer_out);
    layer_in = layer_out;
  }
}

/* Pool */
layer_out = net_get_param(model->list_layer_out, "pool.out")->data;

if (mode == MODE_TRAIN)
  cache = net_get_cache(model->list_layer_cache, "pool.cache");
else
  cache = NULL;
global_avg_pool_forward(layer_in, cache, layer_out);
```

Overview of the training

- Loss function:
  - Forward pass of the network.
  - Calculate loss and its gradient.
  - Backward pass of the network.
  - /src/net_resnet.c
- Weight update
  - Sgd with momentum
  - In src/solver.c
- Next:
  - How to utilize GPU to do expensive convolution operation.



```
/**
 * Compute loss for a batch of (x,y), do forward/backward, and update
 * gradients*/
status_t resnet_loss(model_t const *model, tensor_t x, label_t const labels[],
                     T *ptr_loss) {
  T loss_classify, loss_reg;
  PDBG("========= Forwarding ==========");
  tensor_t out, dout;

  // Forward
  resnet_forward(model, x);

  // Softmax
  param_t *param_score = net_get_param(model->list_layer_out, "fc.out");
  AWNN_CHECK_NE(NULL, labels);
  out = param_score->data;
  dout = param_score->diff;
  PDBG("========= Softmax ==========");
  AWNN_CHECK_EQ(S_OK,
                loss_softmax(out, labels, &loss_classify, MODE_TRAIN, dout));

  // Backward
  PDBG("========= Backwarding ==========");
  AWNN_CHECK_EQ(S_OK, resnet_backward(model, dout, &loss_reg));

  *ptr_loss = loss_classify + loss_reg;
  return S_OK;
}
```

II. Convolution layer structure in context of GEMM strategy

II. Convolution Forward and backward CPU -> GPU and some optimization.

Part II outline : Convolution Forward and backward

- Role in project
- An ok general GPU strategy
- Forward and backward
    - Batched padding and padding removal
        - CPU
        - GPU
    - Batched tensor transposes
        - CPU
        - GPU
    - Im2col and col2im
        - CPU
        - GPU

Part II outline : Convolution Forward and backward

- **My role in the project**
- An ok general GPU strategy
- Forward and backward
  - Batched padding and padding removal
    - CPU
    - GPU
  - Batched tensor transposes
    - CPU
    - GPU
  - Im2col and col2im
    - CPU
    - GPU

Role in project : Chris Goebel

- I worked primarily on the forward and backward functions in both the CPU and GPU.
    - Translation from python to C
    - Translation from C to CUDA

- Also helped a small amount with the framework.

- Wrote a lot of tests.
    - We have 100’s of tests overall.
    - Build Controlled by CMake
    - GPU parameter search

Part II outline :

- Role in project
- **An ok general GPU strategy**
- Forward and backward
  - Batched padding and padding removal
    - CPU
    - GPU
  - Batched tensor transposes
    - CPU
    - GPU
  - Im2col and col2im
    - CPU
    - GPU

A Decent General GPU Strategy

- After the CPU version is complete and tests have been written.
    - Create a mapping from your problem to a 1D access pattern

    - Create the GPU code as a grid stride loop from global index to the 1D mapping.

        - Works for almost any problem with for loops
        - Is sometimes optimal, but mostly not, but is still moderately fast

- Create a harness for your CPU tests

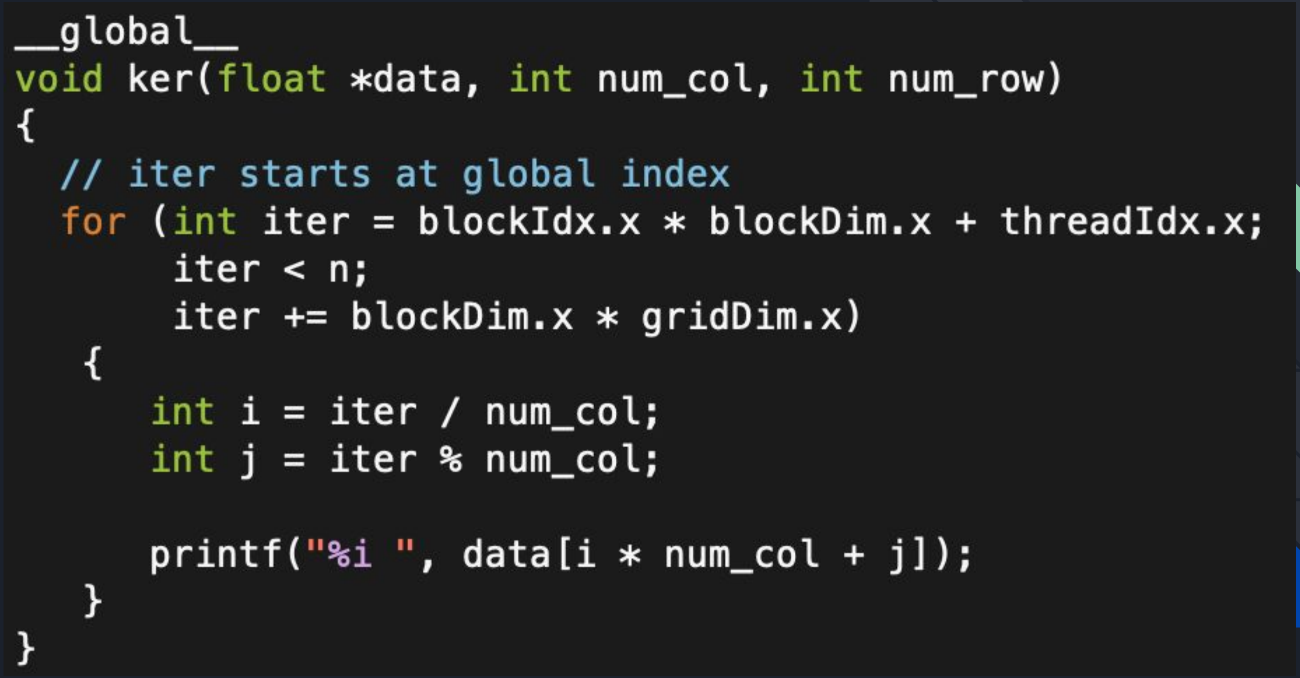
A Decent General GPU Strategy

Example of mapping 2D operation to grid stride loop.

SEE DEMO : 1D_2D_1D_map.cpp

A Decent General GPU Strategy

- On the GPU



```
__global__
void ker(float *data, int num_col, int num_row)
{
  // iter starts at global index
  for (int iter = blockIdx.x * blockDim.x + threadIdx.x;
           iter < n;
           iter += blockDim.x * gridDim.x)
   {
      int i = iter / num_col;
      int j = iter % num_col;

      printf("%i ", data[i * num_col + j]);
   }
}
```

An ok generalized GPU strategy

- In the last lecture we learned about grid stride loops.
  - Benefit from highly coalesced access patterns.
- It turns out you can basically turn anything with a for loop into one of these.
- Relatively easy (not always, as we will see) to implement.
- Generally very good acceleration
  - Often excellent with localization
  - Not always though

Part II outline :

- Role in project
- An ok general GPU strategy
- **Forward and backward**
  - **Batched padding and padding removal**
    - **CPU**
    - **GPU**
  - Batched tensor transposes
    - CPU
    - GPU
  - Im2col and col2im
    - CPU
    - GPU

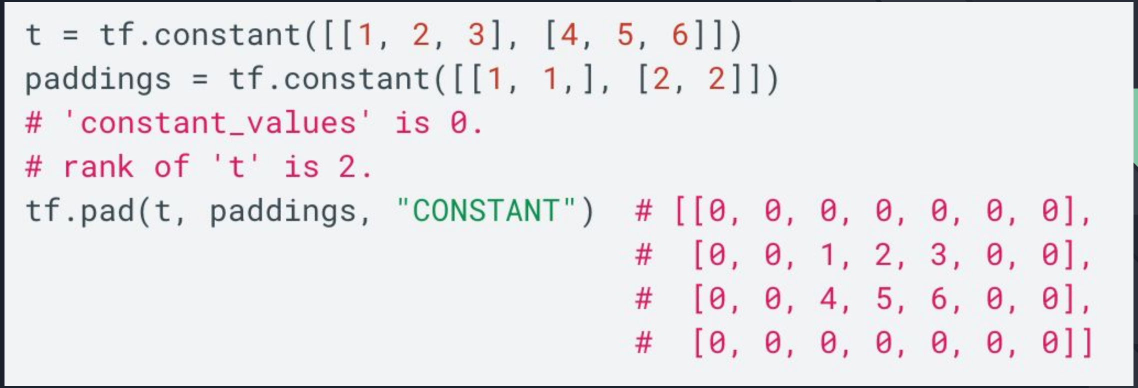
Batched padding and padding removal

- Not a necessary operation, but complexity of the im2col and col2im loops dominated the work.

- General idea is to take a 4D object and transform the lowest 2 dimensions by adding a boundary.

- Is a 1 to 1 mapping, so it fits into the grid stride conversion very well.
    - Some divergence in the cuda kernel because of boundary conditions
    - Can either deal with this here, or deal with the divergence in im2col and col2im.

Add / remove padding memory access.



```
t = tf.constant([[1, 2, 3], [4, 5, 6]])
paddings = tf.constant([[1, 1,], [2, 2]])
# 'constant_values' is 0.
# rank of 't' is 2.
tf.pad(t, paddings, "CONSTANT")  # [[0, 0, 0, 0, 0, 0, 0],
                                 #  [0, 0, 1, 2, 3, 0, 0],
                                 #  [0, 0, 4, 5, 6, 0, 0],
                                 #  [0, 0, 0, 0, 0, 0, 0]]
```

- A very nice visualization. Not exactly what we are dealing with because we are flattening, but shows padding over 3D.

Padding has to work over all channels and has to account for multiple images

- add padding
- remove padding

Part II outline :

- Role in project
- An ok general GPU strategy
- Forward and backward
  - Batched padding and padding removal
    - CPU
    - GPU
  - Batched tensor transposes
    - CPU
    - GPU
  - Im2col and col2im
    - CPU
    - GPU

Batched tensor transpose

- Take a look at this explanation for what the operation does.
  - An explanation of numpy's tensor transpose function.
- Also can look at the source code for numpy
  - Numpy source for transpose operation.

- A complex operation because it requires a complete reorganization in memory.

Batched tensor transpose

- We can notice that the operation is always the same sequence.
- In the forward, we see that the transpose pattern is always 3012, which is a rotation from the least significant dimension to the most.
- After we recognize that we can look at the pattern in memory that the rotation causes.
- Then we can reduce it to a 2D operation.
- However, we in order to enable our generalized strategy (grid stride), we need to turn the operation into a 1D access pattern.
    - Looking at the transpose 3012 code, we can see that the 1D mapping is trivial.
        - But it creates coalesced reads only on one "side" of the mapping.
        - In this case, the write side is coalesced, but the read side is not.
    - The transpose 1230 used by backward is nearly the same.

Part II outline : Convolution Forward and backward

- Role in project
- An ok general GPU strategy
- Forward and backward
  - Batched padding and padding removal
    - CPU
    - GPU
  - Batched tensor transposes
    - CPU
    - GPU
  - Im2col and col2im
    - CPU
    - GPU

col2im and im2col

- Probably the two most important operations because this step actually does the reorganization in preparation for the filters to be applied by GEMM.

- Since we are using cublas for GEMM (and 2D transpose), these functions must be fast.

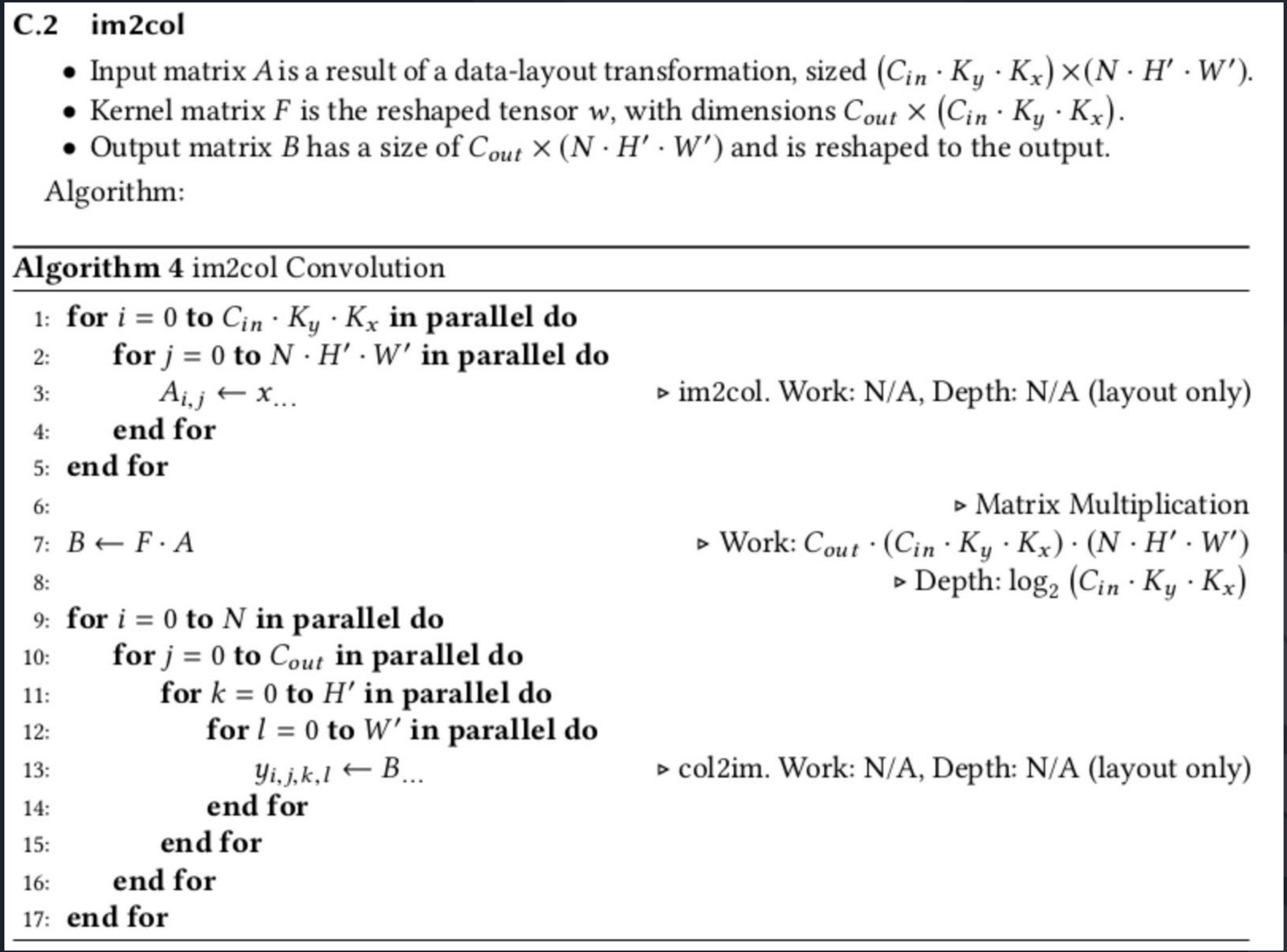
- Both are complex and difficult.
  - im2Col uses a 6D access pattern to map data to a 2D space.
  - col2im expand s2D space back to 4D through a 6D access pattern.

- Of course, these functions were drafted as grid stride kernels.

im2col

- unwraps each location where a filter would be applied into a row in a new 2D matrix
- Is an easy concept, but the filters are applied with a stride, and are not easy to index because they cannot be thought of as always odd, or always square like the book suggests.
- CAFFE EARLY VERSION

im2col



C.2 im2col

- Input matrix A is a result of a data-layout transformation, sized $(C_{in} \cdot K_y \cdot K_x) \times (N \cdot H' \cdot W')$.
- Kernel matrix F is the reshaped tensor w, with dimensions $C_{out} \times (C_{in} \cdot K_y \cdot K_x)$.
- Output matrix B has a size of $C_{out} \times (N \cdot H' \cdot W')$ and is reshaped to the output.

Algorithm:

Algorithm 4 im2col Convolution

```
for $i = 0$ to $C_{in} \cdot K_y \cdot K_x$ in parallel do
    for $j = 0$ to $N \cdot H' \cdot W'$ in parallel do
        $A_{i,j} \leftarrow x_{\dots}$                    ▷ im2col. Work: N/A, Depth: N/A (layout only)
    end for
end for
                                                          ▷ Matrix Multiplication
$B \leftarrow F \cdot A$                                  ▷ Work: $C_{out} \cdot (C_{in} \cdot K_y \cdot K_x) \cdot (N \cdot H' \cdot W')$
                                                          ▷ Depth: $\log_2 (C_{in} \cdot K_y \cdot K_x)$
for $i = 0$ to $N$ in parallel do
    for $j = 0$ to $C_{out}$ in parallel do
        for $k = 0$ to $H'$ in parallel do
            for $l = 0$ to $W'$ in parallel do
                $y_{i,j,k,l} \leftarrow B_{\dots}$        ▷ col2im. Work: N/A, Depth: N/A (layout only)
            end for
        end for
    end for
end for
```

im2col

- Complexity is in the indexing
- Given that our naive strategy is to first try grid strides, we need to turn this into a 1D pattern.

- The naive way to do this is to recognize that the outer 4 dimensions can be collapsed almost exactly like the other 4D to 1D operations we did.
    - However, this leaves each thread to do a lot of work.

- A slightly less naive version can be achieved by collapsing the last two dimensions down to 1D.

im2col

- However, turning this operation into a grid stride loop is not the best approach in general.

- Both sides (read and write) are not coalesced.

- Data reuse occurs, so this is a prime candidate for shared memory usage.
    - Not a candidate for localization because the access locations are so divergent.
    - Would need a localized shared mem map (hash table) to enable it.

col2im

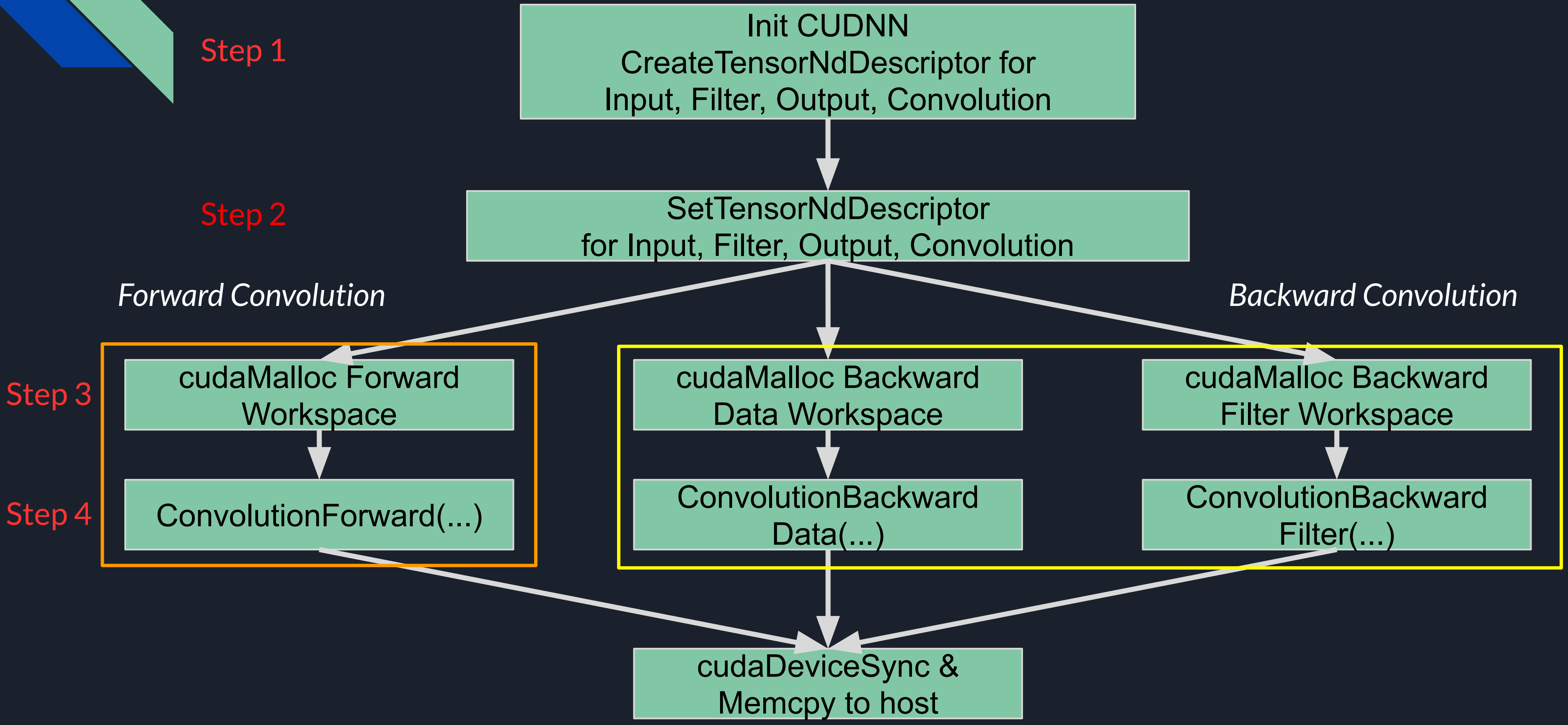
- col2im is almost the same as im2col, except that the addition of a summation requires the use of atomics.
- In this case, atomicAdd with grid stride is not *horrible* because the launch patterns will not cause constant contention.
  - But it is bad
  - Further work needs to be done to optimize both col2im and im2col, but examples for these functions are pretty limited.
    - Caffe (earlier version) code basically does what my naive code does plus the addition of managing the padding in the im2col / col2im.

III. Performance and comparison with CUDNN

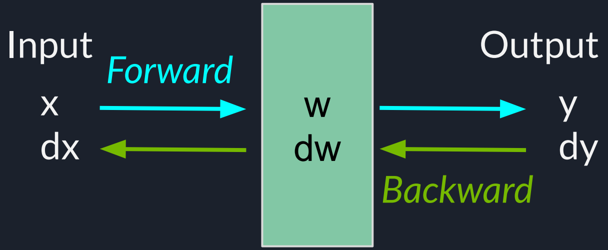
Part III outline :

- Role in project
- Global pooling layer (CPU & GPU)
  - Completed by midterm
- CUDNN convolution implementation within framework
  - Forward & Backward
  - Basic, Interface & notice
  - Improvement
- Experiment performance
  - Different convolution algorithms vs our Naive GPU implementation

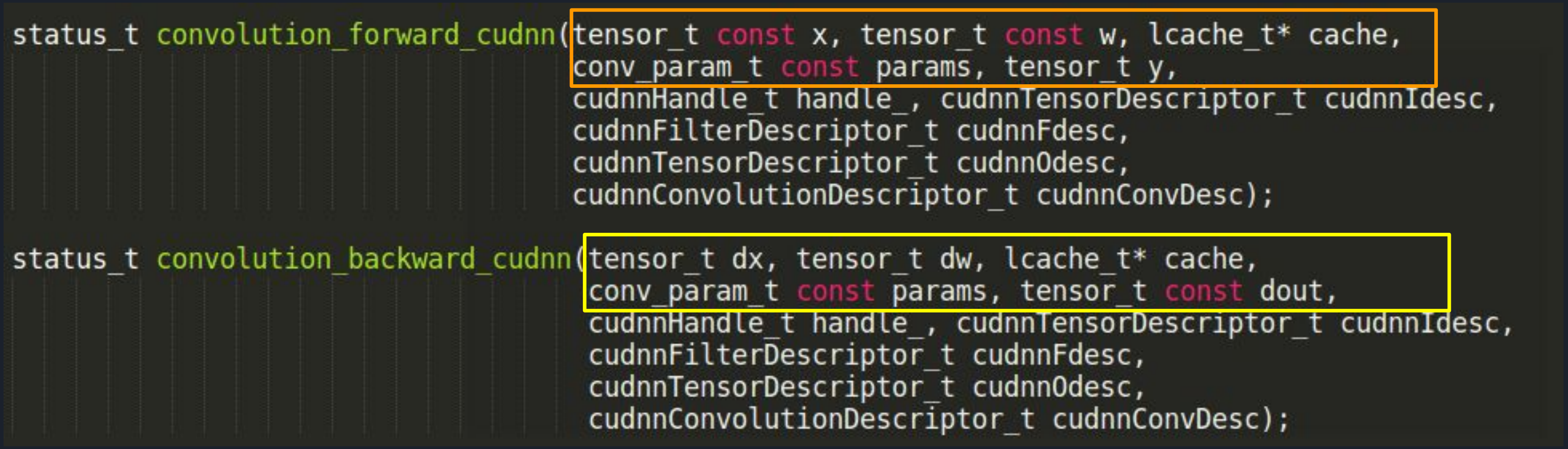
CUDNN Workflow Big Picture


Step 1
Init CUDNN
CreateTensorNdDescriptor for
Input, Filter, Output, Convolution
Step 2
SetTensorNdDescriptor
for Input, Filter, Output, Convolution
Forward Convolution
Backward Convolution
Step 3
cudaMalloc Forward Workspace
cudaMalloc Backward Data Workspace
cudaMalloc Backward Filter Workspace
Step 4
ConvolutionForward(...)
ConvolutionBackward Data(...)
ConvolutionBackward Filter(...)
cudaDeviceSync & Memcpy to host

Cudnn convolution interface

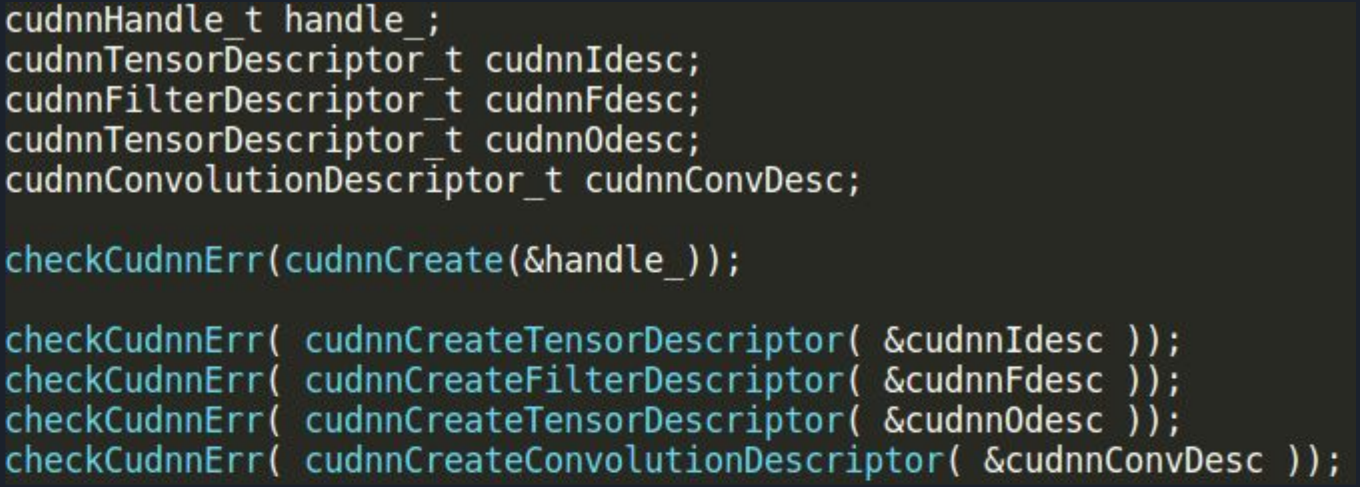


Convolution layer

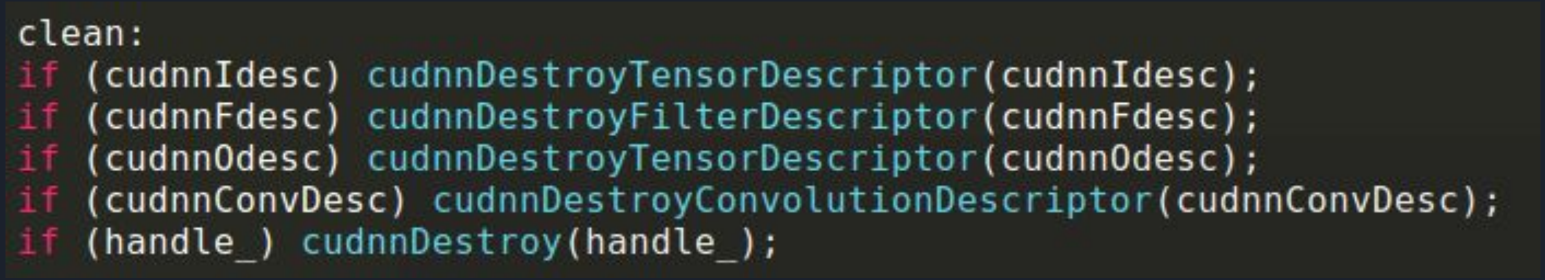


```
status_t convolution_forward_cudnn(tensor_t const x, tensor_t const w, lcache_t* cache,
                                   conv_param_t const params, tensor_t y,
                                   cudnnHandle_t handle_, cudnnTensorDescriptor_t cudnnIdesc,
                                   cudnnFilterDescriptor_t cudnnFdesc,
                                   cudnnTensorDescriptor_t cudnnOdesc,
                                   cudnnConvolutionDescriptor_t cudnnConvDesc);

status_t convolution_backward_cudnn(tensor_t dx, tensor_t dw, lcache_t* cache,
                                    conv_param_t const params, tensor_t const dout,
                                    cudnnHandle_t handle_, cudnnTensorDescriptor_t cudnnIdesc,
                                    cudnnFilterDescriptor_t cudnnFdesc,
                                    cudnnTensorDescriptor_t cudnnOdesc,
                                    cudnnConvolutionDescriptor_t cudnnConvDesc);
```

Step 1: Create Descriptor



```
cudnnHandle_t handle_;
cudnnTensorDescriptor_t cudnnIdesc;
cudnnFilterDescriptor_t cudnnFdesc;
cudnnTensorDescriptor_t cudnnOdesc;
cudnnConvolutionDescriptor_t cudnnConvDesc;

checkCudnnErr(cudnnCreate(&handle_));

checkCudnnErr( cudnnCreateTensorDescriptor( &cudnnIdesc ));
checkCudnnErr( cudnnCreateFilterDescriptor( &cudnnFdesc ));
checkCudnnErr( cudnnCreateTensorDescriptor( &cudnnOdesc ));
checkCudnnErr( cudnnCreateConvolutionDescriptor( &cudnnConvDesc ));
```



```
clean:
if (cudnnIdesc) cudnnDestroyTensorDescriptor(cudnnIdesc);
if (cudnnFdesc) cudnnDestroyFilterDescriptor(cudnnFdesc);
if (cudnnOdesc) cudnnDestroyTensorDescriptor(cudnnOdesc);
if (cudnnConvDesc) cudnnDestroyConvolutionDescriptor(cudnnConvDesc);
if (handle_) cudnnDestroy(handle_);
```

Step 1: Create Descriptor - continue... 1

- cudnnStatus_t cudnnCreate(cudnnHandle_t *handle)
- This function initializes the cuDNN library and creates a handle to an opaque structure holding the cuDNN library context. It *allocates hardware resources on the host and device* and must be called prior to making any other cuDNN library calls.

- cudnnStatus_t cudnnCreateTensorDescriptor(

  cudnnTensorDescriptor_t *tensorDesc)

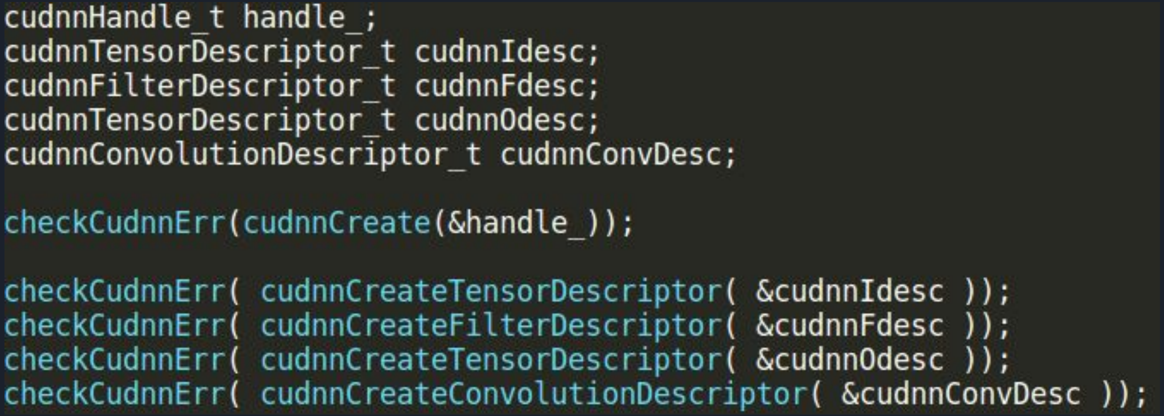
- This function creates a generic tensor descriptor object by allocating the memory needed to hold its opaque structure. The data is initialized to be all zero.

Step 1: Create Descriptor - continue... 2

- cudnnStatus_t cudnnCreateConvolutionDescriptor(

  cudnnConvolutionDescriptor_t *convDesc)

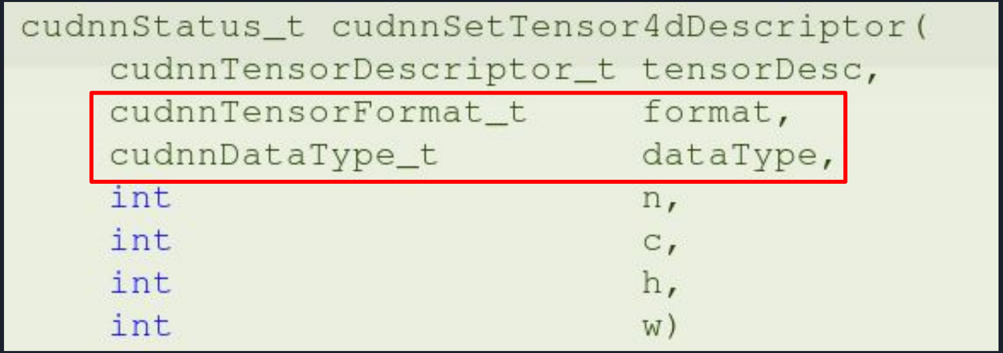
- This function creates a convolution descriptor object by allocating the memory needed to hold its opaque structure.



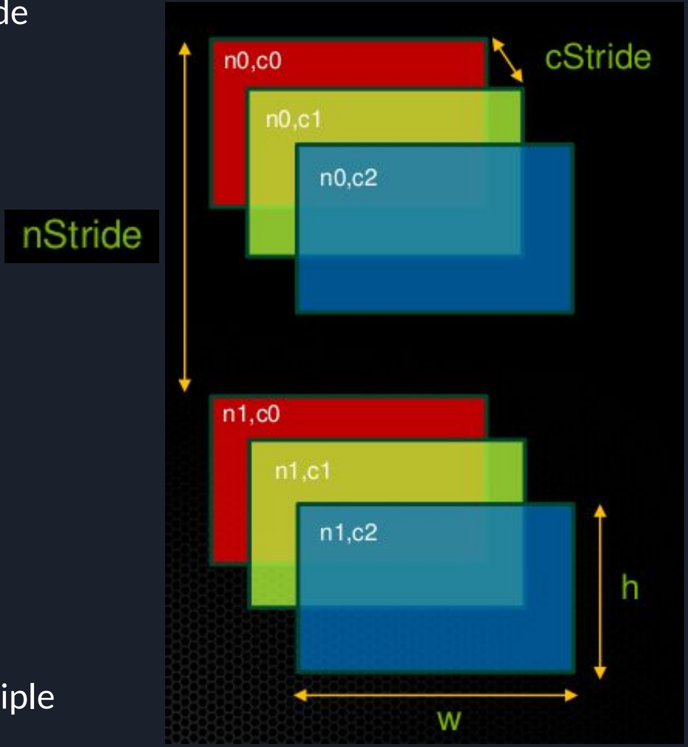
```
cudnnHandle_t handle_;
cudnnTensorDescriptor_t cudnnIdesc;
cudnnFilterDescriptor_t cudnnFdesc;
cudnnTensorDescriptor_t cudnnOdesc;
cudnnConvolutionDescriptor_t cudnnConvDesc;

checkCudnnErr(cudnnCreate(&handle_));

checkCudnnErr( cudnnCreateTensorDescriptor( &cudnnIdesc ));
checkCudnnErr( cudnnCreateFilterDescriptor( &cudnnFdesc ));
checkCudnnErr( cudnnCreateTensorDescriptor( &cudnnOdesc ));
checkCudnnErr( cudnnCreateConvolutionDescriptor( &cudnnConvDesc ));
```

Step 2: Set 4d Tensor for Input & Output

- Image Batches described as 4D Tensor [n, c, h, w] with stride support
  - [nStride, cStride, hStride, wStride]

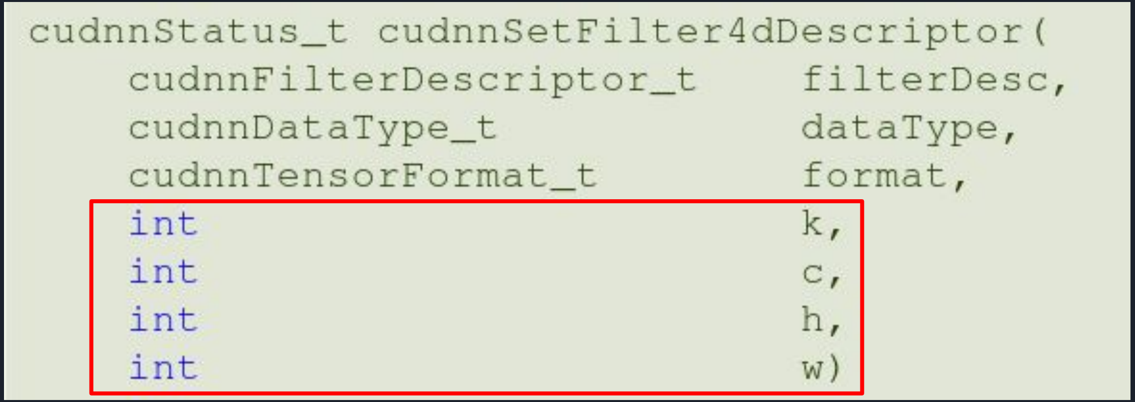


```
cudnnStatus_t cudnnSetTensor4dDescriptor(
    cudnnTensorDescriptor_t tensorDesc,
    cudnnTensorFormat_t     format,
    cudnnDataType_t         dataType,
    int                     n,
    int                     c,
    int                     h,
    int                     w)
```



- format = CUDNN_TENSOR_NCHW
  - CUDNN_TENSOR_NHWC (limited support)
  - CUDNN_TENSOR_NCHW_VECT_C
    - Each element of the tensor is a vector of multiple feature maps
- dataType = CUDNN_DATA_FLOAT
  - CUDNN_DATA_DOUBLE

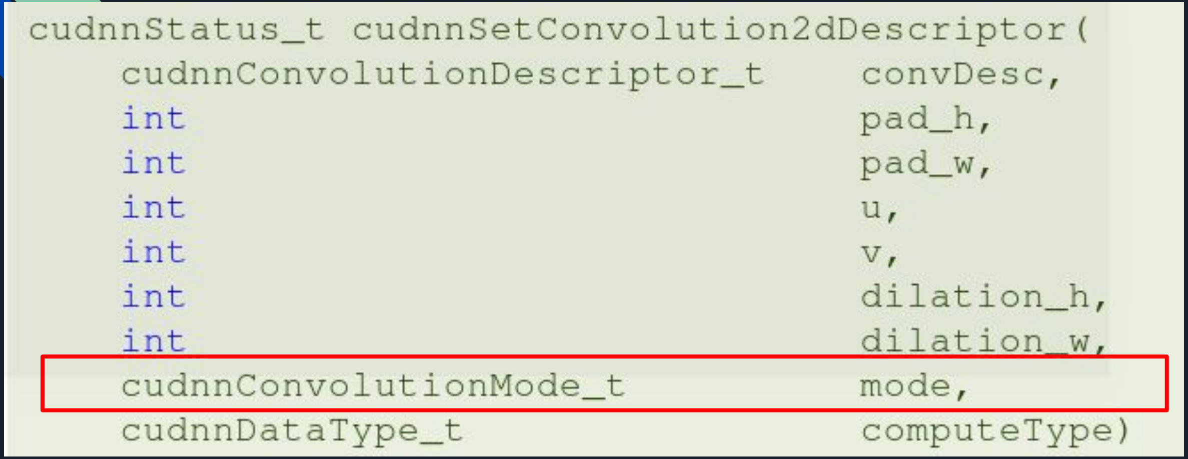
Step 2: Set 4d Filter Descriptor



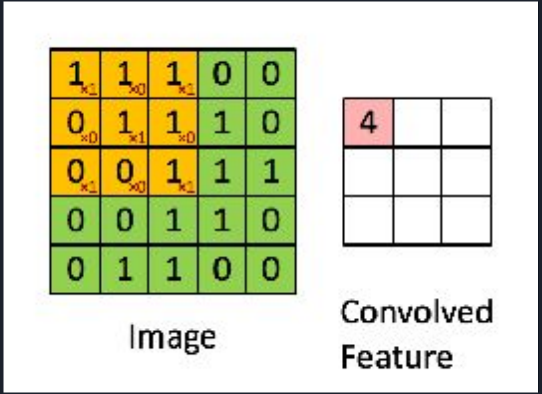
```
cudnnStatus_t cudnnSetFilter4dDescriptor(
    cudnnFilterDescriptor_t    filterDesc,
    cudnnDataType_t            dataType,
    cudnnTensorFormat_t        format,
    int                        k,
    int                        c,
    int                        h,
    int                        w)
```

This function initializes a previously created filter descriptor object. The layout of the filters must be contiguous in memory.

Step 2: Set 2d Convolution layer

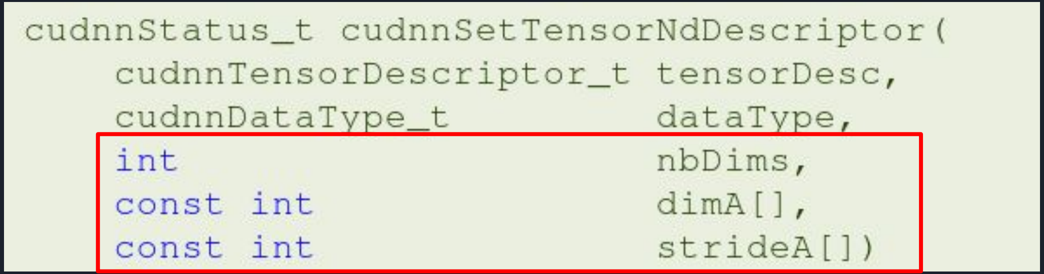


```
cudnnStatus_t cudnnSetConvolution2dDescriptor(
    cudnnConvolutionDescriptor_t    convDesc,
    int                             pad_h,
    int                             pad_w,
    int                             u,
    int                             v,
    int                             dilation_h,
    int                             dilation_w,
    cudnnConvolutionMode_t          mode,
    cudnnDataType_t                 computeType)
```



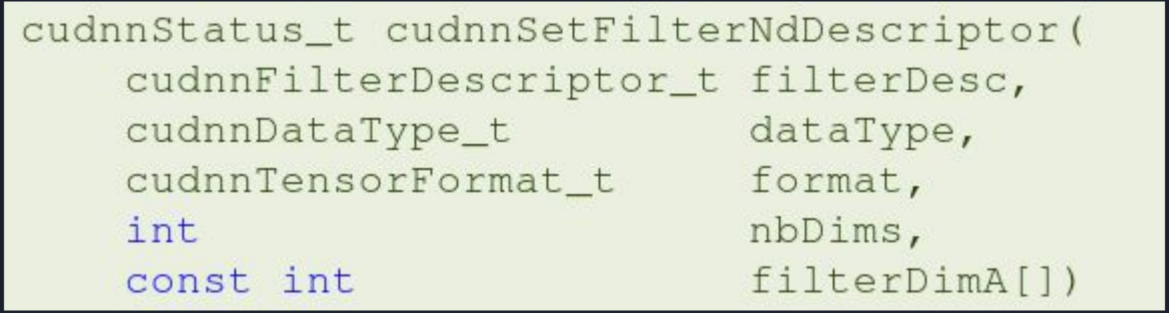
- **u**: Vertical filter stride. **v**: Horizontal filter stride.
- mode = CUDNN_CONVOLUTION and **CUDNN_CROSS_CORRELATION**
  - Original Math convolution with 180 degree flip up → CUDNN_CONVOLUTION
  - Without 180 degree flip up → CUDNN_CROSS_CORRELATION
  - Pass my own verification code, but didn't match framework expected value lists
- **dilation_h** = 1.0, **dilation_w** = 0.0

Set Nd Tensor



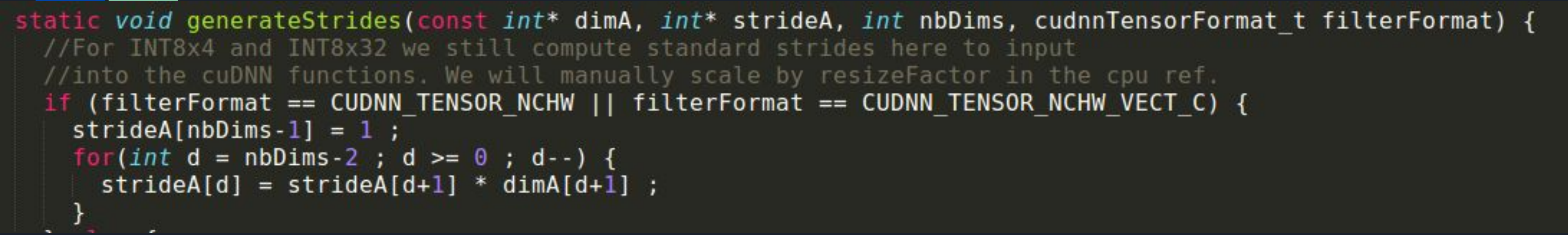
```
cudnnStatus_t cudnnSetTensorNdDescriptor(
    cudnnTensorDescriptor_t tensorDesc,
    cudnnDataType_t         dataType,
    int                     nbDims,
    const int               dimA[],
    const int               strideA[])
```

- nbDims = 4
- dimA[4] ={N, C, H, W}
- **Generate stride**



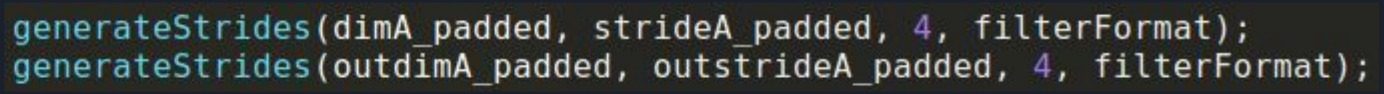
```
cudnnStatus_t cudnnSetFilterNdDescriptor(
    cudnnFilterDescriptor_t filterDesc,
    cudnnDataType_t         dataType,
    cudnnTensorFormat_t     format,
    int                     nbDims,
    const int               filterDimA[])
```

How to generate stride for Nd Tensor



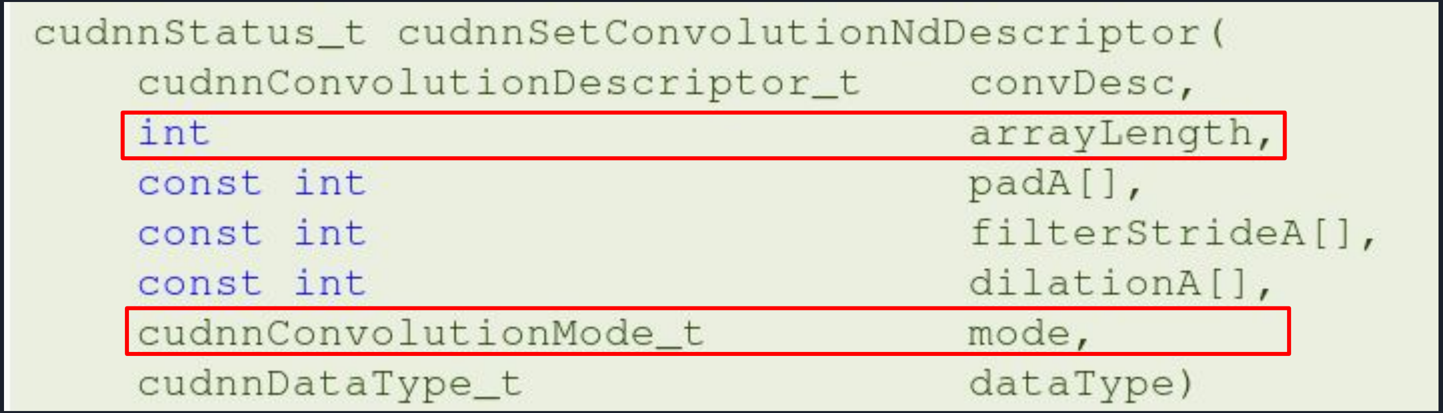
```
static void generateStrides(const int* dimA, int* strideA, int nbDims, cudnnTensorFormat_t filterFormat) {
  //For INT8x4 and INT8x32 we still compute standard strides here to input
  //into the cuDNN functions. We will manually scale by resizeFactor in the cpu ref.
  if (filterFormat == CUDNN_TENSOR_NCHW || filterFormat == CUDNN_TENSOR_NCHW_VECT_C) {
    strideA[nbDims-1] = 1 ;
    for(int d = nbDims-2 ; d >= 0 ; d--) {
      strideA[d] = strideA[d+1] * dimA[d+1] ;
    }
```

- StrideA[3] = 1
- StrideA[2] = dimA[3] = W
- StrideA[1] = dim[2] * strideA[2] = dimA[2] * dimA[3] = H * W
- StrideA[0] = dim[1] * strideA[1] = dimA[1] * dimA[2] * dimA[3] = N * H * W



```
generateStrides(dimA_padded, strideA_padded, 4, filterFormat);
generateStrides(outdimA_padded, outstrideA_padded, 4, filterFormat);
```

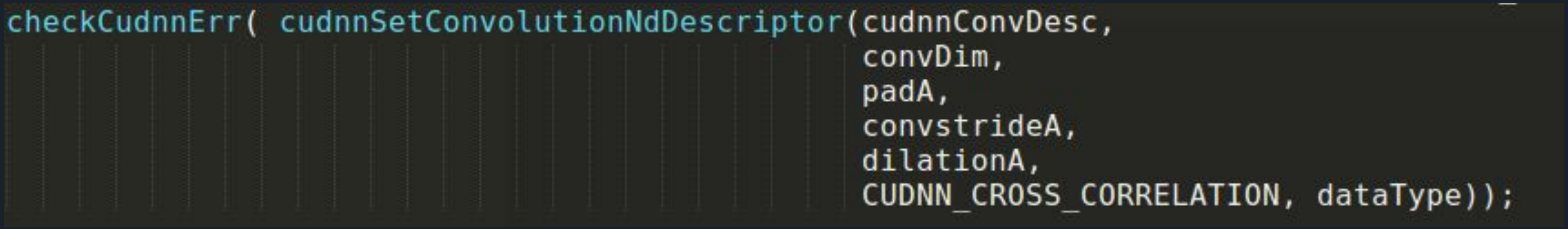
Set Nd Convolution



```
cudnnStatus_t cudnnSetConvolutionNdDescriptor(
    cudnnConvolutionDescriptor_t    convDesc,
    int                             arrayLength,
    const int                       padA[],
    const int                       filterStrideA[],
    const int                       dilationA[],
    cudnnConvolutionMode_t          mode,
    cudnnDataType_t                 dataType)
```

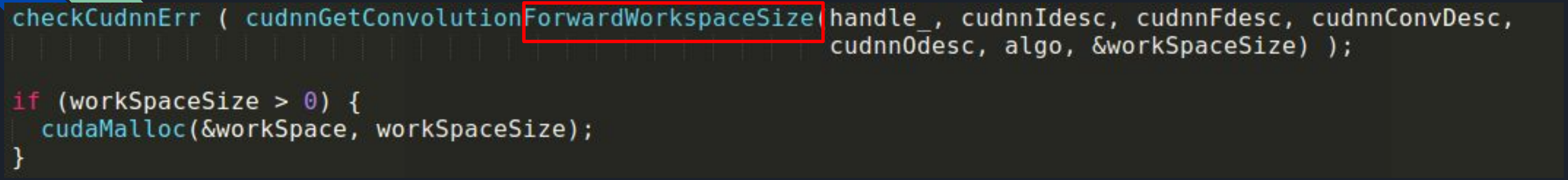
arrayLength = 2 (convDim)

outputDim = 1 + (inputDim + 2*pad - (((filterDim-1)*dilation)+1))/convolutionStride;

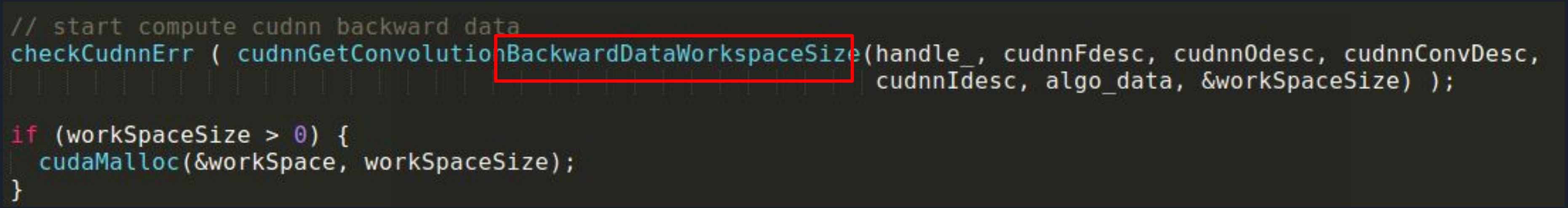


```
checkCudnnErr( cudnnSetConvolutionNdDescriptor(cudnnConvDesc,
                                               convDim,
                                               padA,
                                               convstrideA,
                                               dilationA,
                                               CUDNN_CROSS_CORRELATION, dataType));
```

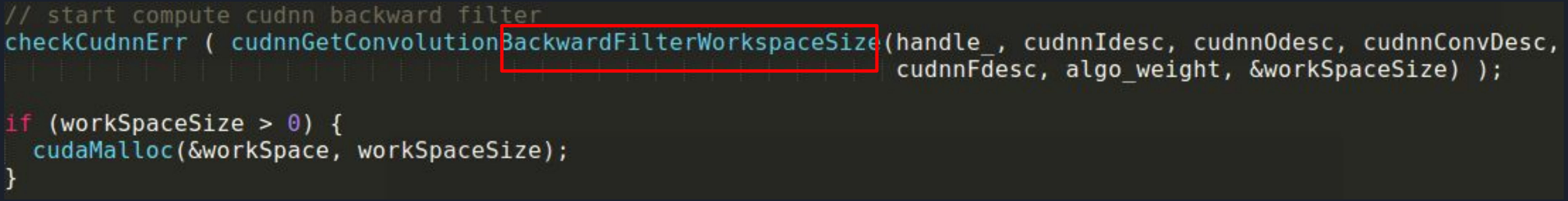
Step 3: Get workspace size & cudaMalloc



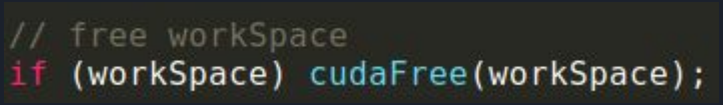
```
checkCudnnErr ( cudnnGetConvolutionForwardWorkspaceSize(handle_, cudnnIdesc, cudnnFdesc, cudnnConvDesc,
                                                        cudnnOdesc, algo, &workSpaceSize) );

if (workSpaceSize > 0) {
  cudaMalloc(&workSpace, workSpaceSize);
}
```



```
// start compute cudnn backward data
checkCudnnErr ( cudnnGetConvolutionBackwardDataWorkspaceSize(handle_, cudnnFdesc, cudnnOdesc, cudnnConvDesc,
                                                             cudnnIdesc, algo_data, &workSpaceSize) );

if (workSpaceSize > 0) {
  cudaMalloc(&workSpace, workSpaceSize);
}
```



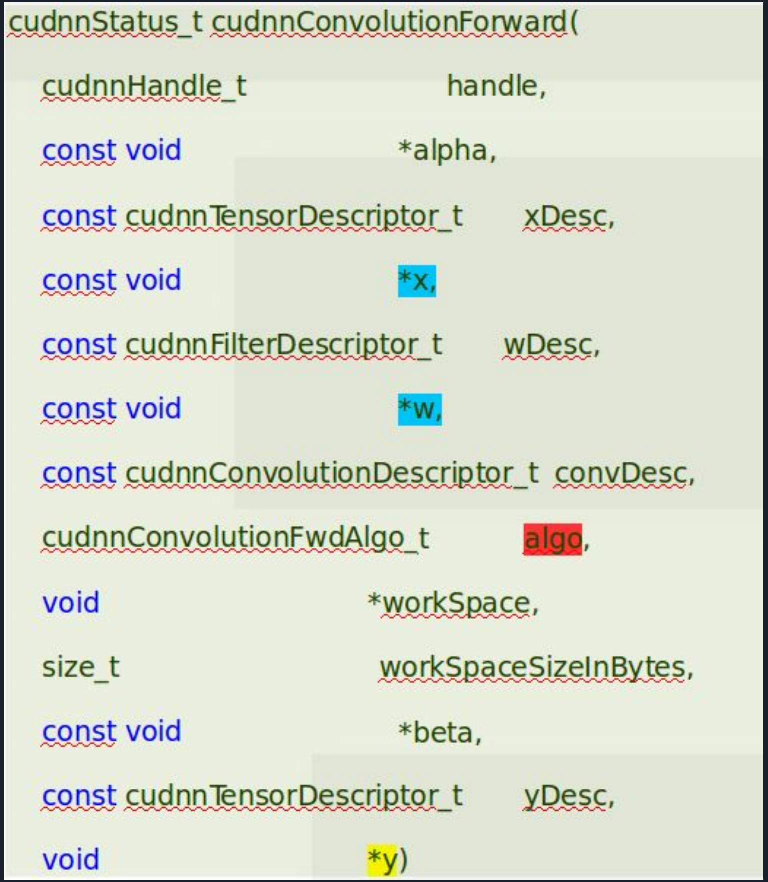
```
// start compute cudnn backward filter
checkCudnnErr ( cudnnGetConvolutionBackwardFilterWorkspaceSize(handle_, cudnnIdesc, cudnnOdesc, cudnnConvDesc,
                                                               cudnnFdesc, algo_weight, &workSpaceSize) );

if (workSpaceSize > 0) {
  cudaMalloc(&workSpace, workSpaceSize);
}
```



```
// free workSpace
if (workSpace) cudaFree(workSpace);
```

Step 4: cudnn Convolution Forward

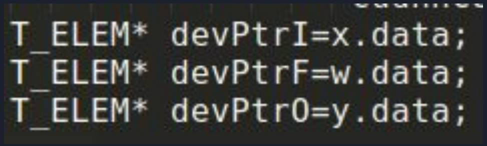
- Executes convolutions or cross-correlations over x using filters specified with w, returning results in y
- Input: x, w
- Output: y
- Algo
    - CUDNN_CONVOLUTION_FWD_ALGO_IMPLICIT_PRECOMP_GEMM
        - *without actually explicitly form the matrix*
    - CUDNN_CONVOLUTION_FWD_ALGO_GEMM
        - Significant workspace may be needed
    - CUDNN_CONVOLUTION_FWD_ALGO_FFT
    - CUDNN_CONVOLUTION_FWD_ALGO_FFT_TILING
    - CUDNN_CONVOLUTION_FWD_ALGO_WINOGRAD
    - CUDNN_CONVOLUTION_FWD_ALGO_WINOGRAD_NONFUSED
        - Significant workspace may be needed



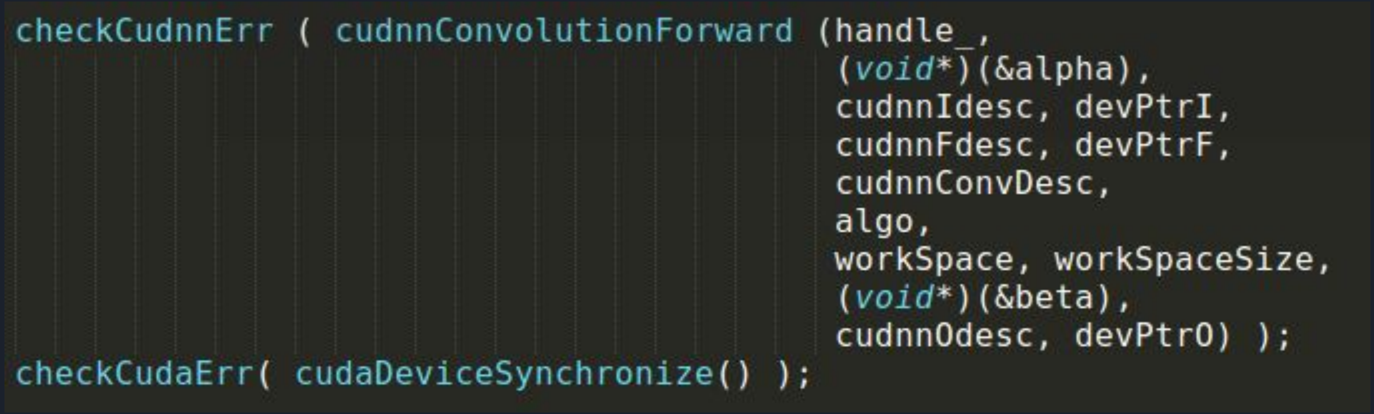
```
cudnnStatus_t cudnnConvolutionForward(
    cudnnHandle_t                      handle,
    const void                         *alpha,
    const cudnnTensorDescriptor_t      xDesc,
    const void                         *x,
    const cudnnFilterDescriptor_t      wDesc,
    const void                         *w,
    const cudnnConvolutionDescriptor_t convDesc,
    cudnnConvolutionFwdAlgo_t          algo,
    void                               *workSpace,
    size_t                             workSpaceSizeInBytes,
    const void                         *beta,
    const cudnnTensorDescriptor_t      yDesc,
    void                               *y)
```

Implementation of Forward

Alpha = 1, beta = 0



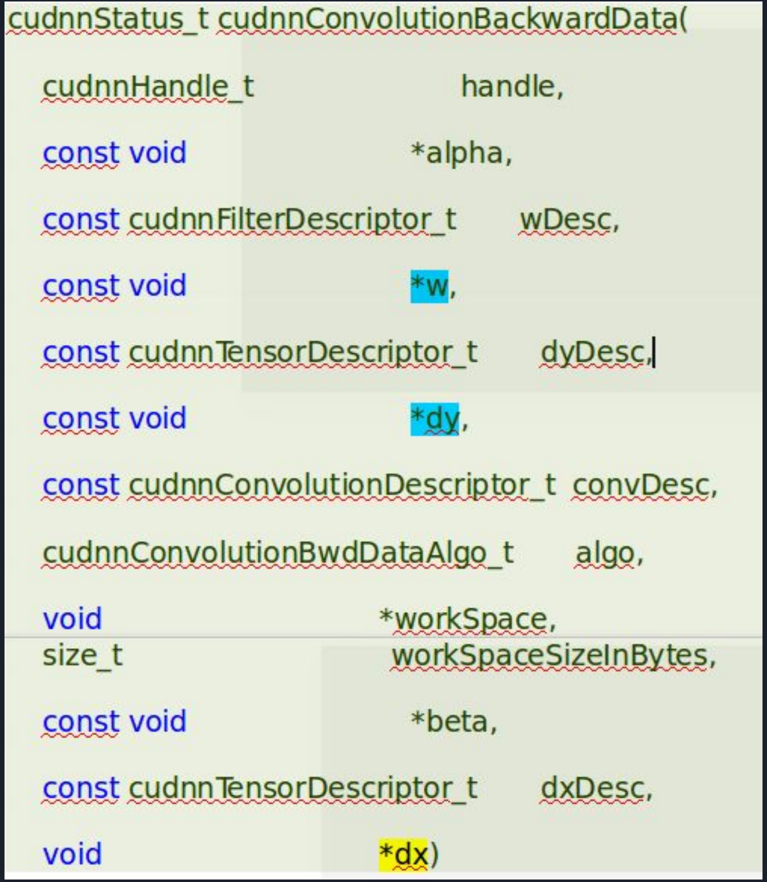
```
T_ELEM* devPtrI=x.data;
T_ELEM* devPtrF=w.data;
T_ELEM* devPtrO=y.data;
```



```
checkCudnnErr ( cudnnConvolutionForward (handle_,
                                        (void*)(&alpha),
                                        cudnnIdesc, devPtrI,
                                        cudnnFdesc, devPtrF,
                                        cudnnConvDesc,
                                        algo,
                                        workSpace, workSpaceSize,
                                        (void*)(&beta),
                                        cudnnOdesc, devPtrO) );
checkCudaErr( cudaDeviceSynchronize() );
```

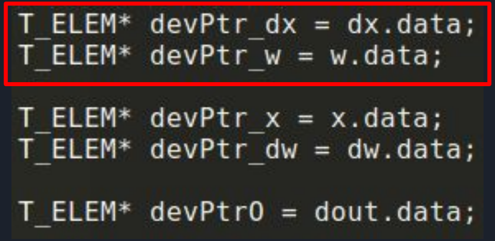
Step 4: cudnn Convolution Backward Data

- Computes the convolution data gradient of the tensor dy
- Input: w, dy
- Output: dx
- Algo
    - CUDNN_CONVOLUTION_BWD_DATA_ALGO_0
        - Non deterministic
    - CUDNN_CONVOLUTION_BWD_DATA_ALGO_1
        - This algorithm expresses the convolution as a matrix product without actually explicitly form the matrix that holds the input tensor data. The results are deterministic.
    - CUDNN_CONVOLUTION_BWD_DATA_ALGO_FFT
    - CUDNN_CONVOLUTION_BWD_DATA_ALGO_FFT_TILING
    - CUDNN_CONVOLUTION_BWD_DATA_ALGO_WINOGRAD
        - Compile error: Undefined enum
    - CUDNN_CONVOLUTION_BWD_DATA_ALGO_WINOGRAD_NONFUSED
        - Compile error: Undefined enum

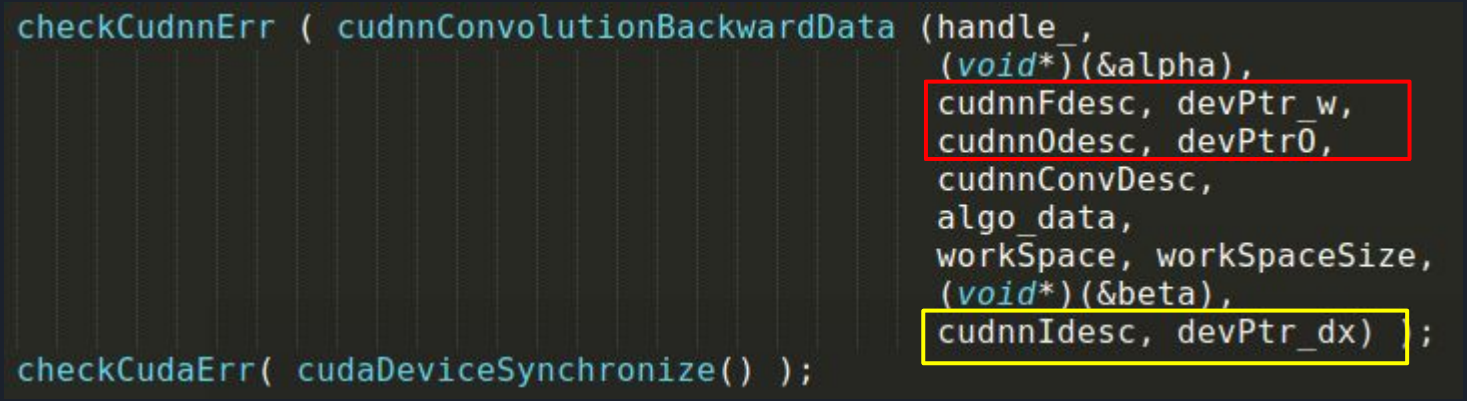


```
cudnnStatus_t cudnnConvolutionBackwardData(
    cudnnHandle_t                    handle,
    const void                      *alpha,
    const cudnnFilterDescriptor_t    wDesc,
    const void                      *w,
    const cudnnTensorDescriptor_t    dyDesc,
    const void                      *dy,
    const cudnnConvolutionDescriptor_t convDesc,
    cudnnConvolutionBwdDataAlgo_t    algo,
    void                            *workSpace,
    size_t                           workSpaceSizeInBytes,
    const void                      *beta,
    const cudnnTensorDescriptor_t    dxDesc,
    void                            *dx)
```

Call Backward convolution data



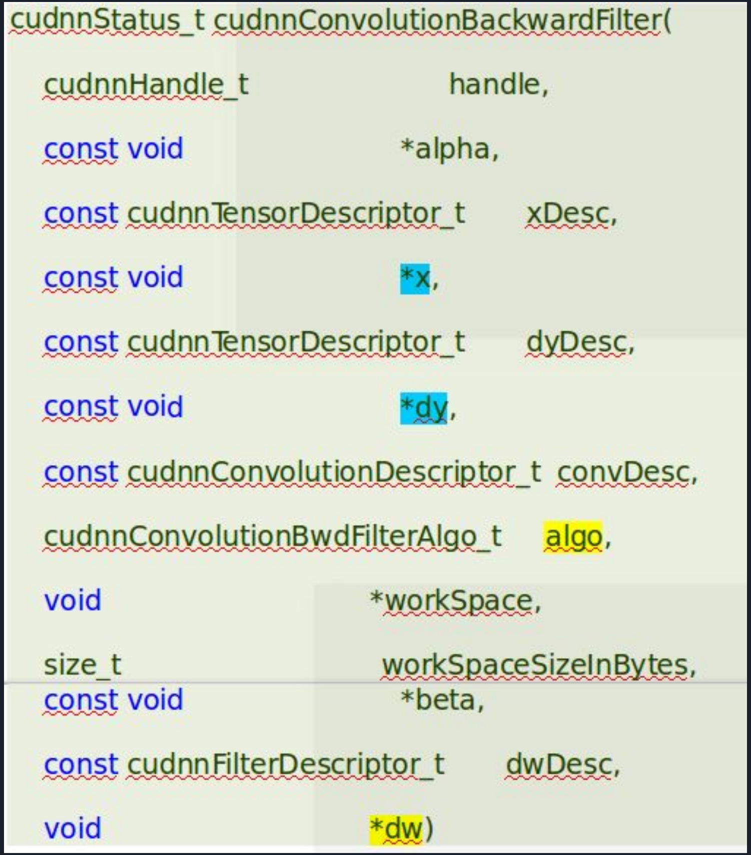
```
T_ELEM* devPtr_dx = dx.data;
T_ELEM* devPtr_w = w.data;

T_ELEM* devPtr_x = x.data;
T_ELEM* devPtr_dw = dw.data;

T_ELEM* devPtrO = dout.data;
```



```
checkCudnnErr ( cudnnConvolutionBackwardData (handle_,
                                               (void*)(&alpha),
                                               cudnnFdesc, devPtr_w,
                                               cudnnOdesc, devPtrO,
                                               cudnnConvDesc,
                                               algo_data,
                                               workSpace, workSpaceSize,
                                               (void*)(&beta),
                                               cudnnIdesc, devPtr_dx) );
checkCudaErr( cudaDeviceSynchronize() );
```

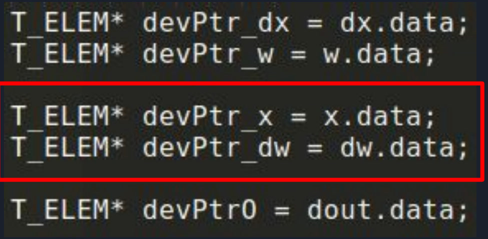
Step 4: cudnn Convolution Backward Filter

- Computes the convolution weight gradient of the tensor dy
- Input: x, dy
- Output: dw
- Algo
  - CUDNN_CONVOLUTION_BWD_FILTER_ALGO_0
    - Non deterministic
  - CUDNN_CONVOLUTION_BWD_FILTER_ALGO_1
    - This algorithm expresses the convolution as a matrix product without actually explicitly form the matrix that holds the input tensor data. The results are deterministic.
  - CUDNN_CONVOLUTION_BWD_FILTER_ALGO_FFT
  - CUDNN_CONVOLUTION_BWD_FILTER_ALGO_FFT_TILING
  - CUDNN_CONVOLUTION_BWD_FILTER_ALGO_WINOGRAD
    - Compile error: Undefined enum
  - CUDNN_CONVOLUTION_BWD_FILTER_ALGO_WINOGRAD_NON FUSED
    - Compile error: Undefined enum

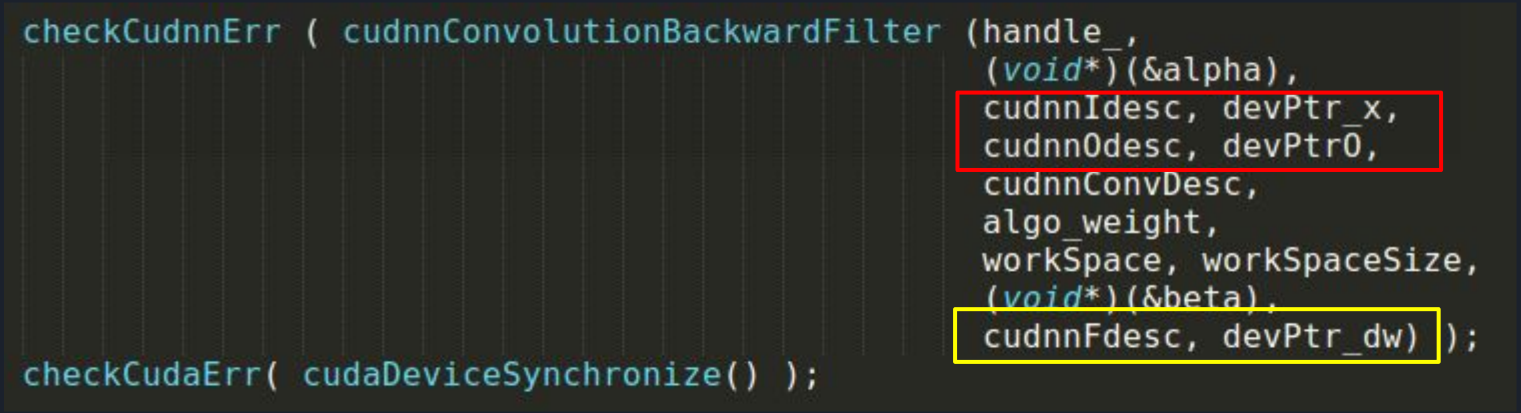


```
cudnnStatus_t cudnnConvolutionBackwardFilter(
    cudnnHandle_t                     handle,
    const void                       *alpha,
    const cudnnTensorDescriptor_t     xDesc,
    const void                       *x,
    const cudnnTensorDescriptor_t     dyDesc,
    const void                       *dy,
    const cudnnConvolutionDescriptor_t convDesc,
    cudnnConvolutionBwdFilterAlgo_t   algo,
    void                             *workSpace,
    size_t                            workSpaceSizeInBytes,
    const void                       *beta,
    const cudnnFilterDescriptor_t     dwDesc,
    void                             *dw)
```

Call Backward convolution filter



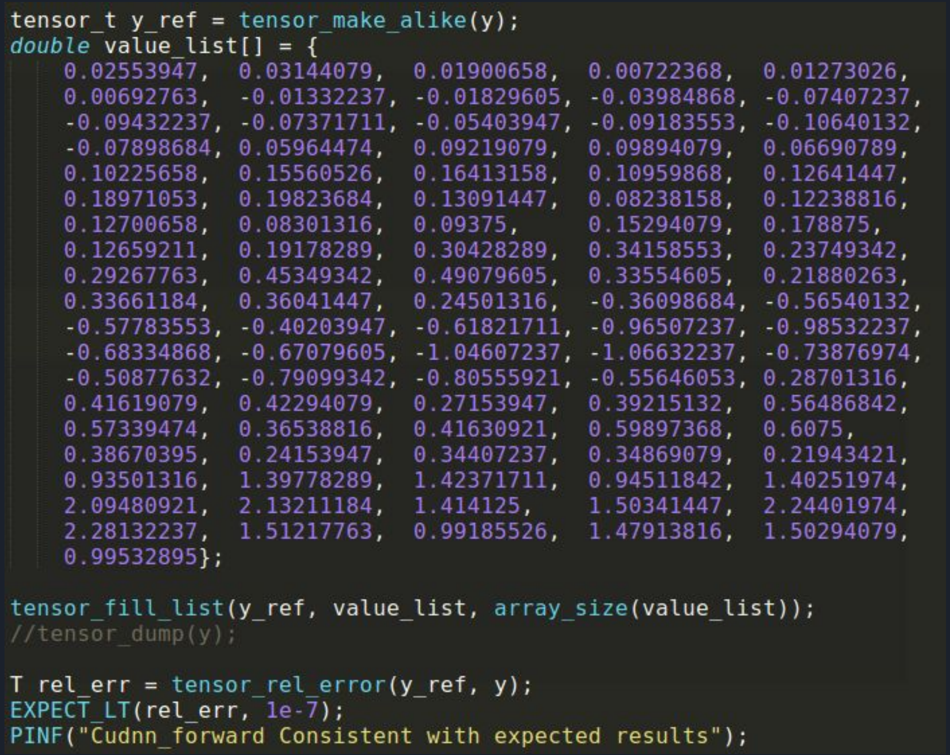
```
T_ELEM* devPtr_dx = dx.data;
T_ELEM* devPtr_w = w.data;

T_ELEM* devPtr_x = x.data;
T_ELEM* devPtr_dw = dw.data;

T_ELEM* devPtrO = dout.data;
```



```
checkCudnnErr ( cudnnConvolutionBackwardFilter (handle_,
                                                 (void*)(&alpha),
                                                 cudnnIdesc, devPtr_x,
                                                 cudnnOdesc, devPtrO,
                                                 cudnnConvDesc,
                                                 algo_weight,
                                                 workSpace, workSpaceSize,
                                                 (void*)(&beta),
                                                 cudnnFdesc, devPtr_dw) );
checkCudaErr( cudaDeviceSynchronize() );
```

Forward Verification

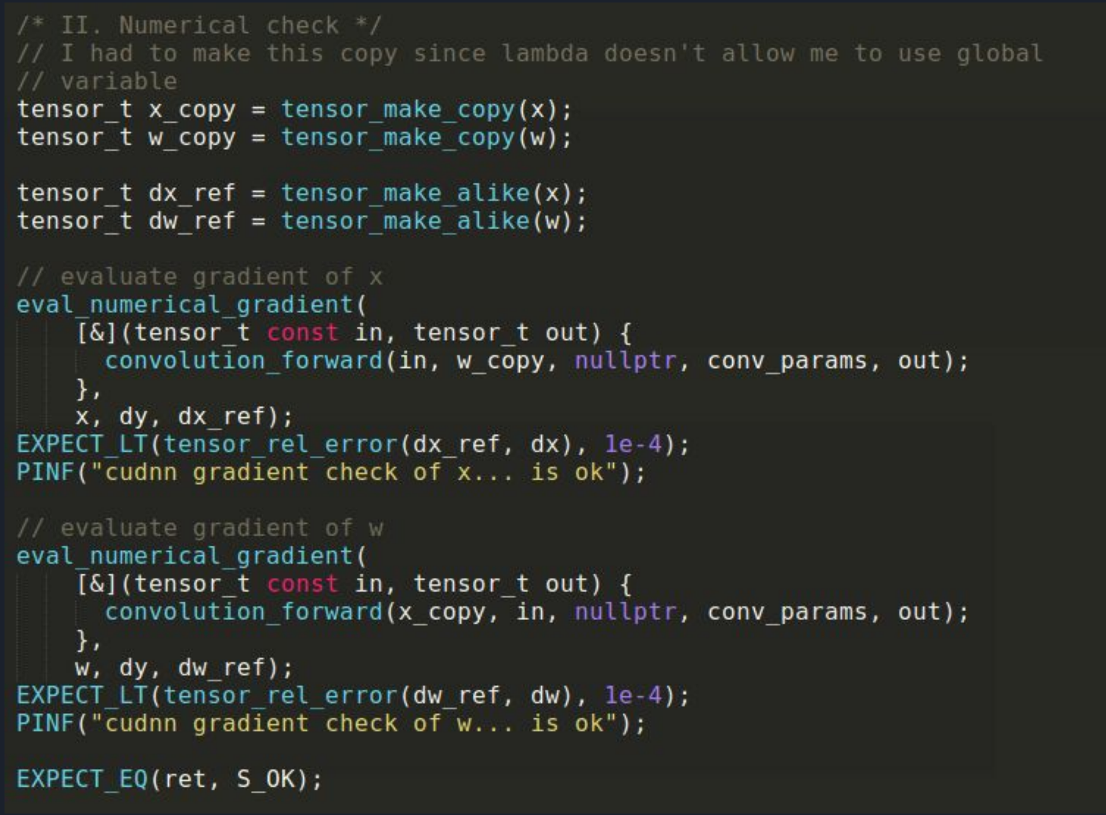
- Google test module
- Relative error Less than 1e-7



```
tensor_t y_ref = tensor_make_alike(y);
double value_list[] = {
    0.02553947,  0.03144079,  0.01900658,  0.00722368,  0.01273026,
    0.00692763,  -0.01332237, -0.01829605, -0.03984868, -0.07407237,
    -0.09432237, -0.07371711, -0.05403947, -0.09183553, -0.10640132,
    -0.07898684, 0.05964474,  0.09219079,  0.09894079,  0.06690789,
    0.10225658,  0.15560526,  0.16413158,  0.10959868,  0.12641447,
    0.18971053,  0.19823684,  0.13091447,  0.08238158,  0.12238816,
    0.12700658,  0.08301316,  0.09375,     0.15294079,  0.178875,
    0.12659211,  0.19178289,  0.30428289,  0.34158553,  0.23749342,
    0.29267763,  0.45349342,  0.49079605,  0.33554605,  0.21880263,
    0.33661184,  0.36041447,  0.24501316,  -0.36098684, -0.56540132,
    -0.57783553, -0.40203947, -0.61821711, -0.96507237, -0.98532237,
    -0.68334868, -0.67079605, -1.04607237, -1.06632237, -0.73876974,
    -0.50877632, -0.79099342, -0.80555921, -0.55646053, 0.28701316,
    0.41619079,  0.42294079,  0.27153947,  0.39215132,  0.56486842,
    0.57339474,  0.36538816,  0.41630921,  0.59897368,  0.6075,
    0.38670395,  0.24153947,  0.34407237,  0.34869079,  0.21943421,
    0.93501316,  1.39778289,  1.42371711,  0.94511842,  1.40251974,
    2.09480921,  2.13211184,  1.414125,    1.50341447,  2.24401974,
    2.28132237,  1.51217763,  0.99185526,  1.47913816,  1.50294079,
    0.99532895};

tensor_fill_list(y_ref, value_list, array_size(value_list));
//tensor_dump(y);

T rel_err = tensor_rel_error(y_ref, y);
EXPECT_LT(rel_err, 1e-7);
PINF("Cudnn_forward Consistent with expected results");
```

Backward Verification

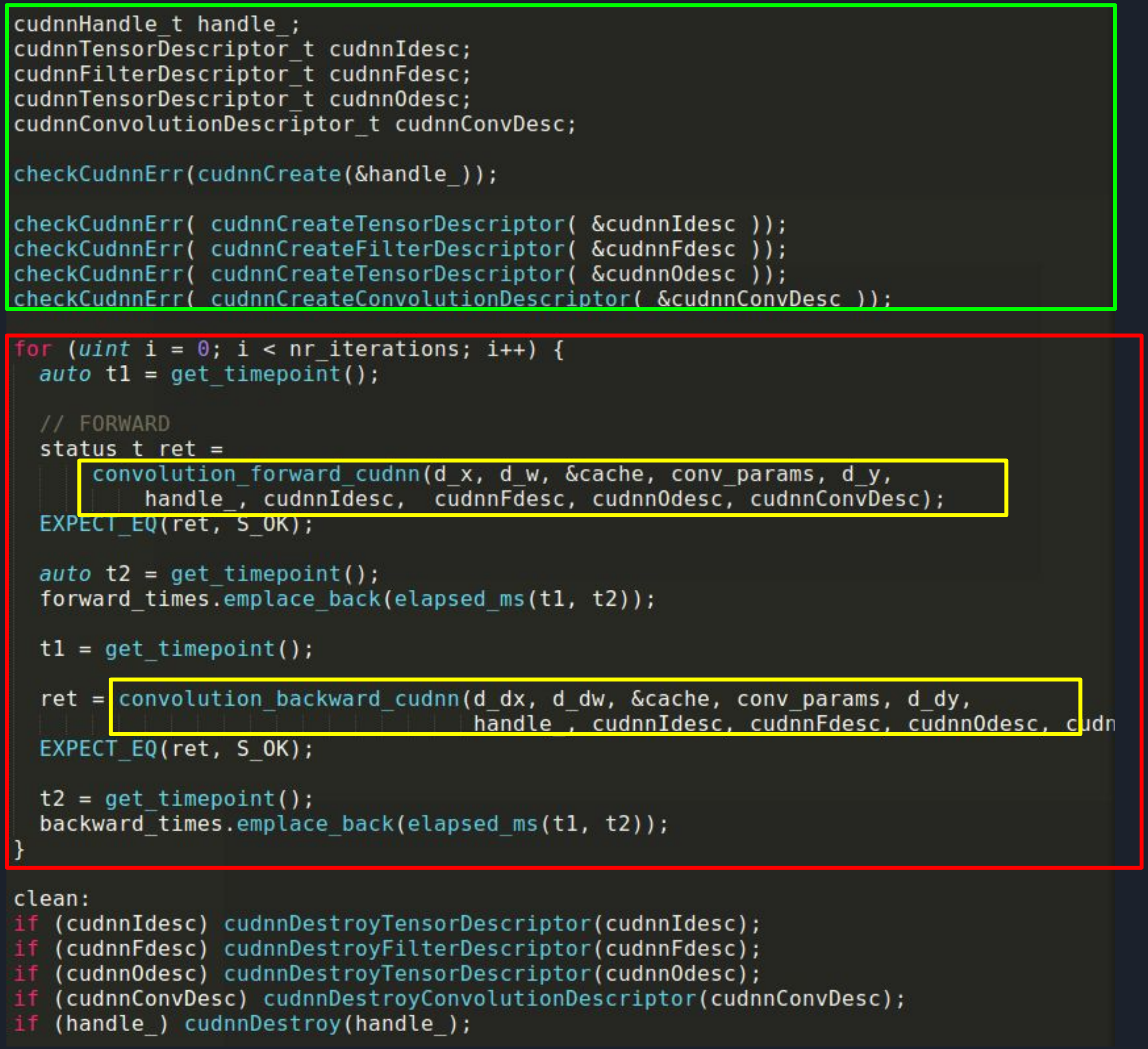
- Numerical test
- Relative Less than 1e-5
  - Data (dx)
  - 5.94404e-05 vs 1e-07
  - Weight (dw)
  - 1.35738e-05 vs 1e-07



```
/* II. Numerical check */
// I had to make this copy since lambda doesn't allow me to use global
// variable
tensor_t x_copy = tensor_make_copy(x);
tensor_t w_copy = tensor_make_copy(w);

tensor_t dx_ref = tensor_make_alike(x);
tensor_t dw_ref = tensor_make_alike(w);

// evaluate gradient of x
eval_numerical_gradient(
    [&](tensor_t const in, tensor_t out) {
      convolution_forward(in, w_copy, nullptr, conv_params, out);
    },
    x, dy, dx_ref);
EXPECT_LT(tensor_rel_error(dx_ref, dx), 1e-4);
PINF("cudnn gradient check of x... is ok");

// evaluate gradient of w
eval_numerical_gradient(
    [&](tensor_t const in, tensor_t out) {
      convolution_forward(x_copy, in, nullptr, conv_params, out);
    },
    w, dy, dw_ref);
EXPECT_LT(tensor_rel_error(dw_ref, dw), 1e-4);
PINF("cudnn gradient check of w... is ok");

EXPECT_EQ(ret, S_OK);
```

Bench test improvement

- Cuda malloc device memory outside
- Create descriptor outside the test
- After Forward, intermediate data are stored in cache
  - x
  - w



```
cudnnHandle_t handle_;
cudnnTensorDescriptor_t cudnnIdesc;
cudnnFilterDescriptor_t cudnnFdesc;
cudnnTensorDescriptor_t cudnnOdesc;
cudnnConvolutionDescriptor_t cudnnConvDesc;

checkCudnnErr(cudnnCreate(&handle_));

checkCudnnErr( cudnnCreateTensorDescriptor( &cudnnIdesc ));
checkCudnnErr( cudnnCreateFilterDescriptor( &cudnnFdesc ));
checkCudnnErr( cudnnCreateTensorDescriptor( &cudnnOdesc ));
checkCudnnErr( cudnnCreateConvolutionDescriptor( &cudnnConvDesc ));
```

```
for (uint i = 0; i < nr_iterations; i++) {
  auto t1 = get_timepoint();

  // FORWARD
  status_t ret =
      convolution_forward_cudnn(d_x, d_w, &cache, conv_params, d_y,
          handle_, cudnnIdesc,  cudnnFdesc, cudnnOdesc, cudnnConvDesc);
  EXPECT_EQ(ret, S_OK);

  auto t2 = get_timepoint();
  forward_times.emplace_back(elapsed_ms(t1, t2));

  t1 = get_timepoint();

  ret = convolution_backward_cudnn(d_dx, d_dw, &cache, conv_params, d_dy,
                                   handle_, cudnnIdesc, cudnnFdesc, cudnnOdesc, cudn
  EXPECT_EQ(ret, S_OK);

  t2 = get_timepoint();
  backward_times.emplace_back(elapsed_ms(t1, t2));
}
```

```
clean:
if (cudnnIdesc) cudnnDestroyTensorDescriptor(cudnnIdesc);
if (cudnnFdesc) cudnnDestroyFilterDescriptor(cudnnFdesc);
if (cudnnOdesc) cudnnDestroyTensorDescriptor(cudnnOdesc);
if (cudnnConvDesc) cudnnDestroyConvolutionDescriptor(cudnnConvDesc);
if (handle_) cudnnDestroy(handle_);
```

Compile & Build

- git clone https://github.com/fenggli/gpu-computing-materials
- mkdir build
- cd build
- cmake ..
- ccmake ..
    - Choose option USE FLOAT32 & CUDA & CUDNN
- make
- ./tests/test-layer-conv-cudnn
- ./tests/bench-conv-cudnn

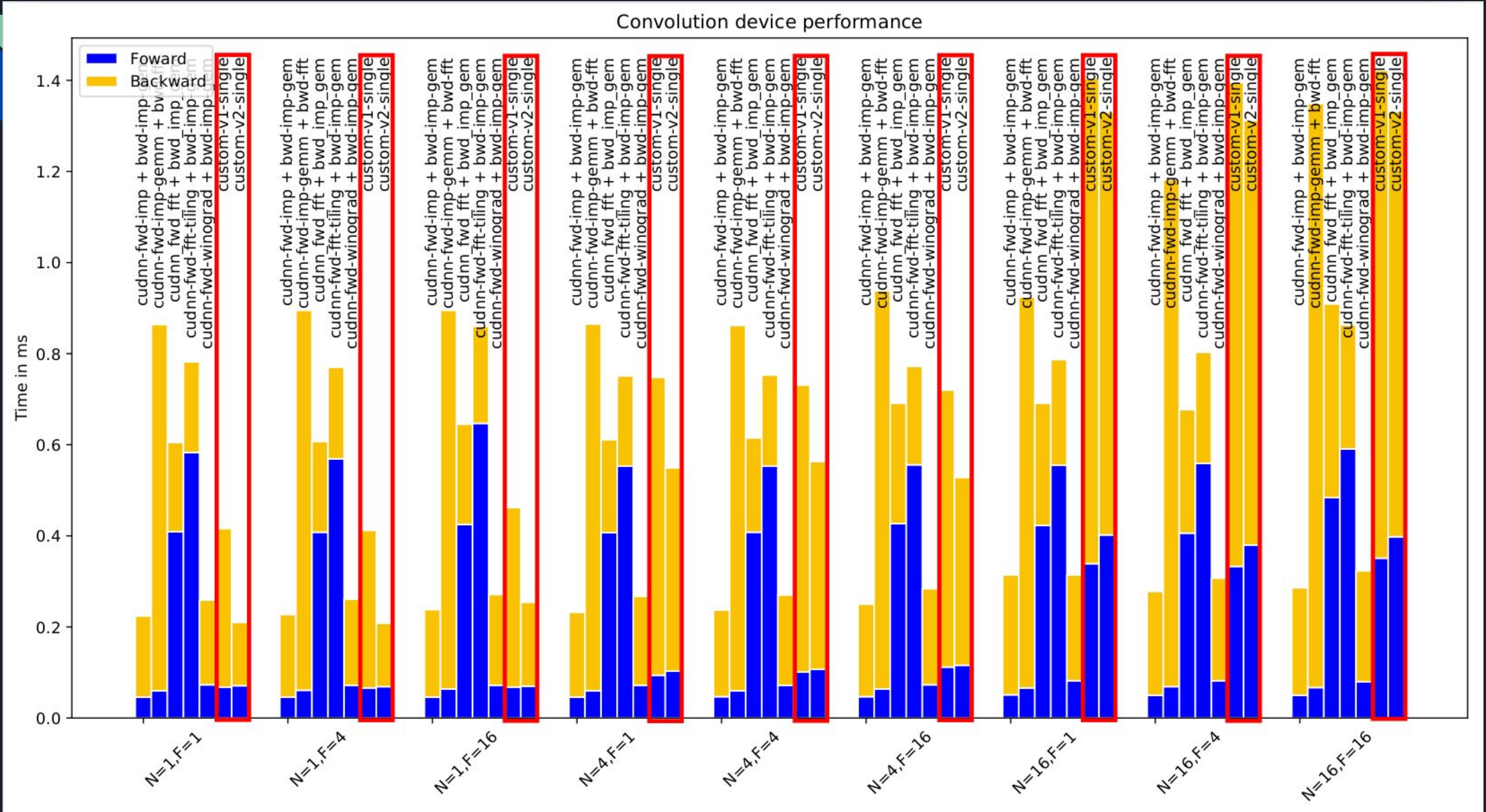
Experiment setup

nr_imgs	nr_input_channel	input h/w	nr_output_ channels	kerne(filter size) h/w	pad	stride
N	C	H and W	F	HH and WW		
1	4	32	1	3	1	1
1	4	32	4	3	1	1
1	4	32	16	3	1	1
4	4	32	1	3	1	1
4	4	32	4	3	1	1
4	4	32	16	3	1	1
16	4	32	1	3	1	1
16	4	32	4	3	1	1
16	4	32	16	3	1	1

- Cudnn
  - Forward: Implicit_gemm / FFT/ FFT tiling/ Winograd
  - Backward: Implicit_gemm / FFT
- Custom
  - V1
  - V2

Performance on Sievert

Using stride =1, pad =1, H=W=32, input channels =4, filter sizes =3x3, 100 iterations



Conclusion

- Framework and Network. (Feng)
- Optimized GPU convolution layer. (Chris)
- Performance and Comparison with CUDNN. (Yuankun)

- A lot more results coming soon!

Q&A

Thanks!