## Interface for Convolutions

To achieve best performance with small convolutions for CNN on SIMD architectures, a specific data layout must be used. As this layout depends on several architectural parameters, the goal of the LIBXSMM's interface is to hide this complexity from the user by providing copy-in and copy-out routines. This happens using opaque data types which themselves are later bound to a convolution operation. The interface is available for C.

The concept of the interface is circled around a few handle types: `libxsmm_dnn_layer`, `libxsmm_dnn_buffer`, `libxsmm_dnn_bias`, and `libxsmm_dnn_filter`. A handle is setup by calling a create-function:

```
/** Simplified LIBXSMM types which are needed to create a handle. */

/** Structure which describes the input and output of data (DNN). */
typedef struct libxsmm_dnn_conv_desc {
  int N;                                  /* number of images in mini-batch */
  int C;                                  /* number of input feature maps */
  int H;                                  /* height of input image */
  int W;                                  /* width of input image */
  int K;                                  /* number of output feature maps */
  int R;                                  /* height of filter kernel */
  int S;                                  /* width of filter kernel */
  int u;                                  /* vertical stride */
  int v;                                  /* horizontal stride */
  int pad_h;                              /* height of logical rim padding to input
                                             for adjusting output height */
  int pad_w;                              /* width of logical rim padding to input
                                             for adjusting output width */
  int pad_h_in;                           /* height of zero-padding in input buffer,
                                             must equal to pad_h for direct conv */
  int pad_w_in;                           /* width of zero-padding in input buffer,
                                             must equal to pad_w for direct conv */
  int pad_h_out;                          /* height of zero-padding in output buffer */
  int pad_w_out;                          /* width of zero-padding in output buffer */
  int threads;                            /* number of threads to use when running
                                             convolution */
  libxsmm_dnn_datatype datatype;          /* datatypes use for all input and outputs */
  libxsmm_dnn_tensor_format buffer_format;  /* format which is for buffer buffers */
  libxsmm_dnn_tensor_format filter_format;  /* format which is for filter buffers */
  libxsmm_dnn_conv_algo algo;             /* convolution algorithm used */
  libxsmm_dnn_conv_option options;        /* additional options */
  libxsmm_dnn_conv_fuse_op fuse_ops;      /* used ops into convolutions */
} libxsmm_dnn_conv_desc;

/** Type of algorithm used for convolutions. */
typedef enum libxsmm_dnn_conv_algo {
  /** let the library decide */
  LIBXSMM_DNN_CONV_ALGO_AUTO,   /* ignored for now */
  /** direct convolution. */
  LIBXSMM_DNN_CONV_ALGO_DIRECT
} libxsmm_dnn_conv_algo;

/** Denotes the element/pixel type of an image/channel. */
typedef enum libxsmm_dnn_datatype {
  LIBXSMM_DNN_DATATYPE_F32,
  LIBXSMM_DNN_DATATYPE_I32,
  LIBXSMM_DNN_DATATYPE_I16,
  LIBXSMM_DNN_DATATYPE_I8
} libxsmm_dnn_datatype;

libxsmm_dnn_layer* libxsmm_dnn_create_conv_layer(
  libxsmm_dnn_conv_desc conv_desc, libxsmm_dnn_err_t* status);
libxsmm_dnn_err_t libxsmm_dnn_destroy_conv_layer(
  const libxsmm_dnn_layer* handle);
```

A sample call looks like (without error checks):

```
/* declare LIBXSMM variables */
libxsmm_dnn_conv_desc conv_desc;
libxsmm_dnn_err_t status;
libxsmm_dnn_layer* handle;
/* setting conv_desc values.... */
```

```
conv_desc.N = ...
/* create handle */
handle = libxsmm_dnn_create_conv_layer(conv_desc, &status);
```

Next activation and filter buffers need to be linked, initialized and bound to the handle. Afterwards the convolution can be executed in a threading environment of choice (error checks are omitted for brevity):

```
float *input, *output, *filter;
libxsmm_dnn_buffer* libxsmm_reg_input;
libxsmm_dnn_buffer* libxsmm_reg_output;
libxsmm_dnn_filter* libxsmm_reg_filter;

/* allocate data */
input = (float*)libxsmm_aligned_malloc(...);
output = ...;

/* link data to buffers */
libxsmm_reg_input = libxsmm_dnn_link_buffer(  libxsmm_handle, LIBXSMM_DNN_INPUT, input,
                                              LIBXSMM_DNN_TENSOR_FORMAT_LIBXSMM_PTR, &status);
libxsmm_reg_output = libxsmm_dnn_link_buffer( libxsmm_handle, LIBXSMM_DNN_OUTPUT, output,
                                              LIBXSMM_DNN_TENSOR_FORMAT_LIBXSMM_PTR, &status);
libxsmm_reg_filter = libxsmm_dnn_link_filter( libxsmm_handle, LIBXSMM_DNN_FILTER, filter,
                                              LIBXSMM_DNN_TENSOR_FORMAT_LIBXSMM_PTR, &status);

/* copy in data to LIBXSMM format: naive format is: */
/* (mini-batch)(number-featuremaps)(featuremap-height)(featuremap-width) for layers, */
/* and the naive format for filters is: */
/* (number-output-featuremaps)(number-input-featuremaps)(kernel-height)(kernel-width) */
libxsmm_dnn_copyin_buffer(libxsmm_reg_input, (void*)naive_input, LIBXSMM_DNN_TENSOR_FORMAT_NCHW);
libxsmm_dnn_zero_buffer(libxsmm_reg_output);
libxsmm_dnn_copyin_filter(libxsmm_reg_filter, (void*)naive_filter, LIBXSMM_DNN_TENSOR_FORMAT_KCRS);

/* bind layer to handle */
libxsmm_dnn_bind_input_buffer(libxsmm_handle, libxsmm_reg_input, LIBXSMM_DNN_REGULAR_INPUT);
libxsmm_dnn_bind_output_buffer(libxsmm_handle, libxsmm_reg_output, LIBXSMM_DNN_REGULAR_OUTPUT);
libxsmm_dnn_bind_filter(libxsmm_handle, libxsmm_reg_filter, LIBXSMM_DNN_REGULAR_FILTER);

/* allocate and bind scratch */
scratch = libxsmm_aligned_scratch(libxsmm_dnn_get_scratch_size(
  libxsmm_handle, LIBXSMM_DNN_COMPUTE_KIND_FWD, &status), 2097152);
libxsmm_dnn_bind_scratch(libxsmm_handle, LIBXSMM_DNN_COMPUTE_KIND_FWD, scratch);

/* run the convolution */
#pragma omp parallel
{
  libxsmm_dnn_convolve_st(libxsmm_handle, LIBXSMM_DNN_CONV_KIND_FWD, 0,
    omp_get_thread_num(), omp_get_num_threads());
}

/* copy out data */
libxsmm_dnn_copyout_buffer(libxsmm_output, (void*)naive_libxsmm_output,
  LIBXSMM_DNN_TENSOR_FORMAT_NCHW);

/* clean up */
libxsmm_dnn_release_scratch(...);
libxsmm_dnn_release_buffer(...);
...
libxsmm_dnn_destroy_buffer(...);
...
libxsmm_dnn_destroy_conv_layer(...);
```