AXILOG

SRS-006-CONVOLUTION

Deterministic 2D Convolution

Draft v0.1 L2 [D1]

Last updated 20 January 2026
Compliance ISO 26262 · IEC 62304 · MISRA-C:2012
Changelog 1 entry

Contents

1. Purpose 3
2. Requirements 3
   2.1 Functional Requirements 3
   2.2 Performance Requirements 5
3. Common Kernel Types 6
   3.1 Edge Detection Kernels 6
   3.2 Smoothing Kernels 6
   3.3 Neural Network Kernels 7
4. Padding Modes (Future) 7
   4.1 Valid Padding (Current Implementation) 7
   4.2 Same Padding (Planned) 7
   4.3 Full Padding (Planned) 7
5. Design Decisions 8
   Why Start With Valid Padding? 8
   Row-Major vs Column-Major 8
6. Verification Criteria 8
7. Performance Characteristics 10
8. Medical Imaging Example 10
   Use Case: Tumor Detection 10
9. Automotive Example 11
   Use Case: Pedestrian Detection 11
10. Implementation 11
11. Usage Example 12

12. Integration with Neural Networks 12
13. Future Extensions 13
14. References 13
15. Revision History 13

Component: Core / Vision **Compliance:** ISO 26262 · IEC 62304 · MISRA-C:2012 **Applicability:** Computer Vision (Medical Imaging, Autonomous Vehicles)

---

1. Purpose

This module defines deterministic 2D convolution operations for computer vision applications. Convolution is the fundamental operation in Convolutional Neural Networks (CNNs) used for image analysis, object detection, and medical diagnostics.

Critical Requirement: The system must perform sliding-window dot products with bit-perfect parity and bounded execution time across all platforms.

---

2. Requirements

2.1 Functional Requirements

SRS-006.1: Valid Padding Convolution

The system shall implement 2D convolution with “valid” padding (no zero-padding).

Mathematical Definition:

```
For input I (H×W), kernel K (KH×KW):
Output O dimensions: (H - KH + 1) × (W - KW + 1)

O[r][c] = Σ(i=0 to KH-1) Σ(j=0 to KW-1) I[r+i][c+j] × K[i][j]
```

Rationale:

- Valid padding has predictable output dimensions
- No ambiguity about boundary handling
- Simplest to verify for certification

Verification: Unit tests confirm correct output dimensions and values.

SRS-006.2: Sliding Window Implementation

The convolution shall use explicit nested loops for the sliding window, with no data-dependent branching in inner loops.

Implementation Pattern:

```
for each output position (r, c):
    accumulator = 0
    for each kernel position (kr, kc):
        accumulator += input[r+kr][c+kc] * kernel[kr][kc]
    output[r][c] = quantize(accumulator)
```

Rationale:

- Explicit loops make WCET analysis tractable
- No hidden optimizations that vary by platform
- Cache behavior is predictable

Verification: Timing analysis confirms O(OH × OW × KH × KW) complexity.

SRS-006.3: 64-bit Accumulation

Intermediate accumulation shall use 64-bit integers to prevent overflow.

Rationale:

- Kernel sizes commonly 3×3 to 11×11
- Maximum accumulation: 11×11 = 121 products
- Each product: 32-bit × 32-bit = 64-bit
- Sum of 121 products requires > 32 bits

Implementation: Same accumulator pattern as SRS-003 (matrix multiplication).

Verification: Overflow tests with maximum-valued kernels and inputs.

SRS-006.4: Quantization Consistency

Output quantization shall use round-to-nearest with explicit rounding constant.

Implementation:

```
accumulator += FIXED_HALF;
output = (fixed_t)(accumulator >> FIXED_SHIFT);
```

Rationale:

- Consistent with SRS-002 and SRS-003
- Minimizes quantization error
- Deterministic across platforms

Verification: Quantization tests verify rounding behavior.

2.2 Performance Requirements

SRS-006.5: Bounded Execution Time

Convolution execution time shall depend only on matrix dimensions, not on data values.

Time Complexity:

```
T = O(OH × OW × KH × KW)
Where:
  OH, OW = Output height, width
  KH, KW = Kernel height, width
```

Rationale:

- Required for real-time systems
- Enables WCET analysis
- Prevents timing side-channels

Verification: Timing tests with various data patterns show identical execution time.

SRS-006.6: Memory Efficiency

Convolution shall operate on caller-provided buffers with no dynamic allocation.

Memory Pattern:

```
Input:  Caller allocates H × W elements
Kernel: Caller allocates KH × KW elements
Output: Caller allocates (H-KH+1) × (W-KW+1) elements
```

Rationale:

- Predictable memory footprint
- No heap fragmentation
- Suitable for embedded systems

Verification: Memory profiling confirms O(1) space complexity.

---

3. Common Kernel Types

3.1 Edge Detection Kernels

Sobel Vertical (detects vertical edges):

```
[-1  0  1]
[-2  0  2]
[-1  0  1]
```

Sobel Horizontal (detects horizontal edges):

```
[-1 -2 -1]
[ 0  0  0]
[ 1  2  1]
```

Laplacian (detects all edges):

```
[ 0  1  0]
[ 1 -4  1]
[ 0  1  0]
```

3.2 Smoothing Kernels

Gaussian Blur (3×3, σ=1):

```
[1  2  1]    (divided by 16)
[2  4  2]
[1  2  1]
```

Box Blur (3×3):

```
[1  1  1]    (divided by 9)
[1  1  1]
[1  1  1]
```

3.3 Neural Network Kernels

LeNet-5 First Layer: 5×5 kernels (learned weights)

ResNet First Layer: 7×7 kernels (learned weights)

MobileNet Depthwise: 3×3 kernels (efficient for mobile)

---

4. Padding Modes (Future)

4.1 Valid Padding (Current Implementation)

- No padding added
- Output smaller than input
- **Status:** ✅ Implemented

4.2 Same Padding (Planned)

- Zero-padding added to maintain size
- Output same dimensions as input
- **Status:** 🔄 Future extension

4.3 Full Padding (Planned)

- Maximum padding
- Output larger than input
- **Status:** 🔄 Future extension

5. Design Decisions

Why Start With Valid Padding?

Valid Padding Chosen:

- ✅ Simplest to implement
- ✅ No ambiguity about boundaries
- ✅ Easiest to verify
- ✅ Common in many architectures

Same Padding Deferred:

- ⏳ Requires zero-padding logic
- ⏳ More complex boundary conditions
- ⏳ Can be added later without breaking API

Row-Major vs Column-Major

Row-Major Chosen:

- ✅ Consistent with SRS-003 (matrix operations)
- ✅ Better cache locality for typical access patterns
- ✅ Matches standard C array layout

6. Verification Criteria

V-006.1: Correctness

Test convolution against known results:

- Simple 3×3 kernel on 5×5 input
- Sobel filters on test patterns
- Compare with reference implementations

Pass Criteria: Bit-identical results.

V-006.2: Determinism

Run same convolution 1000 times:

- Same kernel, same input
- Verify bit-identical outputs

Pass Criteria: All outputs identical.

V-006.3: Platform Independence

Run on multiple architectures:

- x86-64 (Intel/AMD)
- ARM (Cortex-A, Cortex-M)
- RISC-V

Pass Criteria: Bit-identical results across all platforms.

V-006.4: Dimension Validation

Test with mismatched dimensions:

- Kernel larger than input
- Output buffer wrong size
- NULL pointers

Pass Criteria: Safe handling (no crash, predictable behavior).

V-006.5: Overflow Protection

Test with maximum values:

- 11×11 kernel all FIXED_MAX
- Input all FIXED_MAX
- Verify no overflow

Pass Criteria: 64-bit accumulator handles all cases.

7. Performance Characteristics

Time Complexity:

```
Valid Padding: O(OH × OW × KH × KW)

Example:
  Input: 224×224
  Kernel: 3×3
  Output: 222×222
  Operations: 222 × 222 × 3 × 3 = 443,556 MACs
```

Space Complexity:

```
O(1) - all buffers caller-provided
No temporary allocations
Stack usage: < 100 bytes
```

Cache Behavior:

- Row-major layout optimizes cache usage
- Spatial locality in sliding window
- Predictable access patterns

8. Medical Imaging Example

Use Case: Tumor Detection

Input: 512×512 grayscale CT scan slice

Layer 1: 5×5 edge detection kernels (4 filters)

- Output: 508×508×4 feature maps
- Detects tumor boundaries

Layer 2: 3×3 refinement kernels (8 filters)

- Output: 506×506×8 feature maps
- Classifies tissue types

Requirement: Bit-perfect results across all hospital scanners.

Our Value: Same CT scan → Same feature maps → Same diagnosis, regardless of hardware.

---

9. Automotive Example

Use Case: Pedestrian Detection

Input: 640×480 camera frame (cropped to ROI)

Layer 1: 7×7 edge kernels (16 filters)

- Detects human silhouettes

Layer 2: 5×5 shape kernels (32 filters)

- Identifies body parts

Layer 3: 3×3 refinement kernels (64 filters)

- Final classification

Requirement: ISO 26262 ASIL-D certification requires deterministic behavior.

Our Value: Prove car's vision system behaves identically in all conditions.

---

10. Implementation

Files:

- `include/convolution.h` - API specification
- `src/core/convolution.c` - Implementation
- `tests/unit/test_convolution.c` - Verification
- `examples/edge_detection.c` - Demonstration

Traceability:

- Code: `@traceability SRS-006-CONVOLUTION`
- Tests: Link to this document in header

11. Usage Example

```
/* Edge detection with Sobel filter */

// Input image (8×8 grayscale)
fixed_t input_buf[64];
fx_matrix_t input;
fx_matrix_init(&input, input_buf, 8, 8);
//  ... fill with image data

// Sobel vertical kernel (3×3)
fixed_t kernel_buf[9] = {
    fixed_from_int(-1), fixed_from_int(0), fixed_from_int(1),
    fixed_from_int(-2), fixed_from_int(0), fixed_from_int(2),
    fixed_from_int(-1), fixed_from_int(0), fixed_from_int(1)
};
fx_matrix_t kernel;
fx_matrix_init(&kernel, kernel_buf, 3, 3);

// Output (6×6)
fixed_t output_buf[36];
fx_matrix_t output;
fx_matrix_init(&output, output_buf, 6, 6);

// Run convolution
fx_conv2d(&input, &kernel, &output);

// Result: vertical edges detected in output
```

12. Integration with Neural Networks

Typical CNN Layer:

```
// Conv2D → BatchNorm → ReLU

fx_conv2d(&input, &weights, &conv_out);      // Convolution
fx_matrix_add_bias(&conv_out, &bias);        // Add bias
// (BatchNorm - future)
fx_relu(&conv_out);                          // Activation
```

Multi-Channel Convolution (Future):

- Current: Single-channel (grayscale)

- Planned: Multi-channel (RGB, feature maps)
- Extension: Depth-wise separable convolutions

13. Future Extensions

SRS-006.7: (Planned) Same Padding Support

SRS-006.8: (Planned) Multi-Channel Convolution

SRS-006.9: (Planned) Strided Convolution

SRS-006.10: (Planned) Dilated Convolution

14. References

- **MISRA-C:2012** - Rule 17.7 (Return value checking)
- **ISO 26262-6:2018** - Software unit design
- **IEC 62304:2006** - Medical device software
- **LeCun et al. (1998)** - "Gradient-Based Learning Applied to Document Recognition"
- **Krizhevsky et al. (2012)** - "ImageNet Classification with Deep CNNs"

15. Revision History

Version	Date	Author	Changes
1.0	2026-01-15	William Murray	Initial version

Document Classification: Technical Specification
Approval Status: Approved for Implementation
Next Review: 2026-04-15