# 3D Point Cloud Network Configurations

This document provides detailed network architectures and configurations for 5 state-of-the-art 3D sparse convolution networks used in LiDAR point cloud processing.

## Overview

All networks are designed for semantic segmentation of 3D point clouds with the following standard configuration: - **Input**: 100,000 points with 4 channels (x, y, z, intensity) - **Output**: 20 semantic classes - **Voxel size**: 0.05m resolution - **Sparsity modeling**: Advanced spatial, feature, weight, and channel sparsity

## Sparsity Configuration

### Default Sparsity Parameters

```
@dataclass
class SparsityConfig:
    spatial_sparsity: float = 0.05    # 5% spatial occupancy (LiDAR typical)
    feature_sparsity: float = 0.5     # 50% feature sparsity (ReLU zeros)
    weight_sparsity: float = 0.3      # 30% weight sparsity (conservative pruning)
    channel_sparsity: float = 0.0     # No channel pruning
```

### Dataset-Specific Spatial Sparsity

- **SemanticKITTI**: 3-5% voxel occupancy
- **nuScenes**: 8-12% voxel occupancy

- **ScanNet (indoor)**: 15-25% voxel occupancy
- **S3DIS (indoor)**: 20-30% voxel occupancy

### Weight Sparsity Potential

| Network | Conservative (<1% acc loss) | Moderate (<3% acc loss) | Aggressive (<5% acc loss) |
|---|---|---|---|
| MinkowskiNet | 30% | 50% | 70% |
| SPVNAS | 25% | 45% | 65% |
| LargeKernel3D | 20% | 40% | 60% |
| VoxelNeXt | 25% | 45% | 65% |
| RSN | 30% | 50% | 70% |

---

## 1. MinkowskiNet

**Paper**: 4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks
**Repository**: https://github.com/NVIDIA/MinkowskiEngine

### Architecture Overview

MinkowskiNet follows a U-Net style encoder-decoder architecture with sparse convolutions.

### Network Configuration

#### Initial Convolution

- **Input**: 4 channels → 32 channels
- **Kernel size**: 3×3×3
- **Operation**: Sparse 3D convolution

#### Encoder (Downsampling Path)

| Stage | Input Channels | Output Channels | Stride | Kernel Size | Operation |
|---|---|---|---|---|---|
| 1 | 32 | 64 | 2 | 3×3×3 | Sparse conv + downsample |
| 2 | 64 | 128 | 2 | 3×3×3 | Sparse conv + downsample |
| 3 | 128 | 256 | 2 | 3×3×3 | Sparse conv + downsample |
| 4 | 256 | 512 | 2 | 3×3×3 | Sparse conv + downsample |

#### Decoder (Upsampling Path)

| Stage | Input Channels | Output Channels | Stride | Kernel Size | Operation |
|---|---|---|---|---|---|
| 1 | 512 | 256 | 2 | 3×3×3 | Transposed conv + upsample |
| 2 | 256 | 128 | 2 | 3×3×3 | Transposed conv + upsample |
| 3 | 128 | 64 | 2 | 3×3×3 | Transposed conv + upsample |
| 4 | 64 | 32 | 2 | 3×3×3 | Transposed conv + upsample |

#### Classification Head

- **Input**: 32 channels → 20 classes
- **Kernel size**: 1×1×1
- **Operation**: Point-wise classification

#### Key Features

- Symmetric encoder-decoder design
- Consistent 3×3×3 kernel size throughout
- 8× downsampling in encoder ($2^3$ at each stage)
- Skip connections between encoder and decoder stages

---

## 2. SPVNAS (Sparse Point-Voxel NAS)

**Paper**: Searching Efficient 3D Architectures with Sparse Point-Voxel Convolution
**Repository**: https://github.com/mit-han-lab/spvnas

### Architecture Overview

SPVNAS uses Neural Architecture Search (NAS) to find efficient sparse convolution architectures with optimized channel configurations.

### Network Configuration

#### Stem Convolutions

| Layer | Input Channels | Output Channels | Kernel Size | Operation |
|---|---|---|---|---|
| Stem 1 | 4 | 32 | 3×3×3 | Sparse 3D conv |
| Stem 2 | 32 | 32 | 3×3×3 | Sparse 3D conv |

**SPVNAS Blocks (NAS-Optimized)**

| Block | Input Channels | Output Channels | Kernel Size | Stride | Description |
|---|---|---|---|---|---|
| 1 | 32 | 48 | 3×3×3 | 1 | Efficient expansion |
| 2 | 48 | 64 | 3×3×3 | 2 | Moderate growth + downsample |
| 3 | 64 | 128 | 3×3×3 | 1 | Standard growth |
| 4 | 128 | 256 | 3×3×3 | 2 | Final expansion + downsample |

**Decoder with Skip Connections**

| Stage | Input Channels | Output Channels | Kernel Size | Operation |
|---|---|---|---|---|
| 1 | 256 | 128 | 3×3×3 | Upsample (4×) |
| 2 | 128 | 64 | 3×3×3 | Upsample (4×) |
| 3 | 64 | 32 | 3×3×3 | Upsample (4×) |

**Point-wise Classification**

- **Input**: 32 channels → 20 classes
- **Kernel size**: 1×1×1
- **Operation**: Point-wise sparse convolution

**Key Features**

- NAS-optimized channel progression: 32→48→64→128→256
- Strategic downsampling at blocks 2 and 4
- Efficient 4× upsampling in decoder
- Skip connections for feature fusion

---

## 3. LargeKernel3D

**Paper**: Large Kernel Convolutions for 3D Processing
**Repository**: https://github.com/dvlab-research/LargeKernel3D

### Architecture Overview

LargeKernel3D explores the use of large kernel convolutions (5×5×5, 7×7×7, 9×9×9) to capture extended spatial context in 3D scenes.

### Network Configuration

#### Stem Layer

- **Input**: 4 channels → 64 channels
- **Kernel size**: 3×3×3
- **Operation**: Standard sparse convolution

#### Large Kernel Stages

| Stage | Input Channels | Output Channels | Kernel Size | Stride | Description |
|---|---|---|---|---|---|
| 1 | 64 | 96 | 5×5×5 | 1 | Medium kernel |
| 2 | 96 | 128 | 7×7×7 | 2 | Large kernel + downsample |
| 3 | 128 | 192 | 9×9×9 | 1 | Very large kernel |
| 4 | 192 | 256 | 7×7×7 | 2 | Large kernel + downsample |

#### Decoder with Progressive Kernel Reduction

| Stage | Input Channels | Output Channels | Kernel Size | Operation |
|---|---|---|---|---|
| 1 | 256 | 192 | 5×5×5 | Upsample (4×) |
| 2 | 192 | 128 | 3×3×3 | Upsample (4×) |
| 3 | 128 | 64 | 3×3×3 | Upsample (4×) |

#### Classification Head

- **Input**: 64 channels → 20 classes

- **Kernel size**: 1×1×1
- **Operation**: Point-wise classification

### Key Features

- Progressive kernel size increase: 3→5→7→9→7
- Largest kernel (9×9×9) for maximum receptive field
- Progressive kernel size reduction in decoder
- Balanced channel growth: 64→96→128→192→256

---

## 4. VoxelNeXt

**Paper**: VoxelNeXt: Fully Sparse VoxelNet for 3D Object Detection and Tracking
**Repository**: https://github.com/dvlab-research/VoxelNeXt

### Architecture Overview

VoxelNeXt adapts ConvNeXt design principles to 3D voxel processing with depthwise convolutions and MLP blocks.

### Network Configuration

#### Patchify Operation

- **Input**: 4 channels → 48 channels
- **Kernel size**: 4×4×4
- **Operation**: ConvNeXt-style patchify

#### ConvNeXt-Style Stages

| Stage | Input Channels | Output Channels | Blocks | Kernel Size | Downsampling |
|---|---|---|---|---|---|
| 1 | 48 | 96 | 2 | 3×3×3 | 8× (aggressive) |
| 2 | 96 | 192 | 2 | 3×3×3 | 8× (aggressive) |
| 3 | 192 | 384 | 6 | 3×3×3 | 8× (main stage) |
| 4 | 384 | 768 | 2 | 3×3×3 | 8× (aggressive) |

**ConvNeXt Block Structure** Each block contains: 1. **Depthwise Convolution**: Channel-wise 3×3×3 convolution 2. **MLP Expansion**: 1×1×1 conv with 4× channel expansion 3. **MLP Contraction**: 1×1×1 conv back to original channels

```
Block: DW Conv (3×3×3) → MLP (4× expansion) → MLP (contraction)
```

#### Classification Head

- **Input**: 768 channels → 20 classes
- **Kernel size**: 1×1×1
- **Operation**: Global average pooling + classification

#### Key Features

- ConvNeXt-inspired design for 3D processing
- Aggressive 8× downsampling per stage
- Depthwise separable convolutions for efficiency
- 4× MLP expansion ratio (similar to ConvNeXt)
- Largest channel dimension: 768

---

## 5. RSN (Range Sparse Net)

**Paper**: Range Sparse Net for LiDAR 3D Object Detection
**Repository**: https://github.com/caiyuanhao1998/RSN

### Architecture Overview

RSN is specifically designed for LiDAR processing with range-aware features and efficient skip connections.

### Network Configuration

#### Initial Feature Extraction

- **Input**: 4 channels → 32 channels
- **Kernel size**: 3×3×3
- **Operation**: LiDAR-optimized sparse convolution

#### Range-Aware Encoder

| Stage | Input Channels | Output Channels | Kernel Size | Stride | Features |
|---|---|---|---|---|---|
| 1 | 32 | 64 | 3×3×3 | 2 | Main + residual paths |
| 2 | 64 | 128 | 3×3×3 | 2 | Main + residual paths |

| Stage | Input Channels | Output Channels | Kernel Size | Stride | Features |
|---|---|---|---|---|---|
| 3 | 128 | 256 | 3×3×3 | 2 | Main + residual paths |
| 4 | 256 | 512 | 3×3×3 | 2 | Main + residual paths |

**Range-Aware Decoder with Skip Connections**

| Stage | Input Channels | Output Channels | Kernel Size | Stride | Skip Connection |
|---|---|---|---|---|---|
| 1 | 512 | 256 | 3×3×3 | 2 | From encoder stage 4 |
| 2 | 256 | 128 | 3×3×3 | 2 | From encoder stage 3 |
| 3 | 128 | 64 | 3×3×3 | 2 | From encoder stage 2 |
| 4 | 64 | 32 | 3×3×3 | 2 | From encoder stage 1 |

**Range-Aware Feature Fusion**

- **Input**: 32 channels → 64 channels
- **Kernel size**: 1×1×1
- **Operation**: Multi-scale feature fusion

**Final Classification**

- **Input**: 64 channels → 20 classes
- **Kernel size**: 1×1×1
- **Operation**: Point-wise classification

**Key Features**

- Range-aware processing for LiDAR data
- Dual-path architecture (main + residual)
- Symmetric encoder-decoder with skip connections
- Multi-scale feature fusion
- LiDAR-specific optimizations

---

## Performance Characteristics

### MAC Operations Comparison

The networks show different computational profiles:

1. **Most Efficient**: SPVNAS (NAS-optimized channels)
2. **Balanced**: MinkowskiNet (symmetric design)
3. **Large Receptive Field**: LargeKernel3D (large kernels)
4. **Feature Rich**: VoxelNeXt (deep stages)
5. **Range Optimized**: RSN (LiDAR-specific)

### Memory vs Compute Bound Analysis

- **Early layers**: Memory-bound (gather dominates)
- **Deep layers**: Compute-bound (GEMM dominates)
- **Large kernels**: Higher compute intensity
- **Skip connections**: Additional memory traffic

### Sparsity Utilization

All networks benefit significantly from: - **Spatial sparsity**: 3-7% occupancy reduces computation by ~20× - **Feature sparsity**: ReLU zeros provide additional 2× reduction - **Weight sparsity**: Pruning enables 1.5-3× further reduction - **Combined effect**: Up to 100× total speedup possible

---

## Implementation Notes

### Sparse Convolution Types

1. **Standard Sparse Conv**: Allows sparsity pattern changes
2. **Submanifold Conv**: Maintains input sparsity pattern
3. **Strided Conv**: Downsampling with sparsity handling
4. **Transposed Conv**: Upsampling with sparsity restoration

### Memory Optimization

- **Gather operations**: Collect sparse features
- **Hash tables**: Fast neighbor lookup
- **Memory pooling**: Efficient allocation
- **Gradient checkpointing**: Reduced memory during training

### Hardware Considerations

- **GPU utilization**: Irregular memory access patterns
- **Memory bandwidth**: Gather-scatter operations
- **Cache efficiency**: Spatial locality in 3D data

- **Parallelization**: Thread divergence in sparse operations

| **Model** | **Stage** | **In → Out Channels** | **Kernel Size** | **# Blocks** | **Citation** |
|---|---|---|---|---|---|
| **RSN** | Sparse-CNN 1 | 16 → 32 | $3^3$ | 2 | Sun et al., "RSN: Range Sparse Net…", CVPR 2021 (CVF Open Access) |
| | Sparse-CNN 2 | 32 → 64 | $3^3$ | 2 | |
| | Sparse-CNN 3 | 64 → 128 | $3^3$ | 2 | |
| **Cylinder3D** | Block 1 | 5 → 32 | $3^3$ | 3 | Zhou et al., "Cylinder3D for LiDAR Segmentation", CVPR 2021 (CVF Open Access) |
| | Downsample 1 | 32 → 64 | $3^3$, stride=2 | 1 | |
| | Block 2 | 64 → 64 | $3^3$ | 5 | |
| | Downsample 2 | 64 → 128 | $3^3$, stride=2 | 1 | |
| | Block 3 | 128 → 128 | $3^3$ | 5 | |
| **SPVNAS** | Voxel Conv 1 | 4 → 32 | $3^3$ | 2 | Tang et al., "SPVNAS: Searching Efficient 3D Architectures…", ECCV 2020 (ecva.net) |
| | Voxel Conv 2 | 32 → 64 | $3^3$ | 2 | |
| | Voxel Conv 3 | 64 → 96 | $3^3$ | 3 | |
| | Voxel Conv 4 | 96 → 128 | $3^3$ | 3 | |
| **FocalsConv** | Base Backbone | 64 → [64, 128, 256] | $3^3$ | (3, 5, 5) | Xu et al., "FocalsConv: Bridging Regular & Submanifold Convs", CVPR 2022 (arXiv) |

| **Model** | **Stage** | **In → Out Channels** | **Kernel Size** | **# Blocks** | **Citation** |
|---|---|---|---|---|---|
| | Focal Conv | 64 → 64 / 128 → 128 | learned "focal" | 1 per block | |
| **FSD** | Block 1 | 32 → 64 | $3^3$ | 3 | Fan et al., "Fully Sparse 3D Object Detection (SST)", NeurIPS 2022 (lue.fan) |
| | Downsample 1 | 64 → 128 | $3^3$, stride=2 | 1 | |
| | Block 2 | 128 → 256 | $3^3$ | 5 | |
| | Downsample 2 | 256 → 512 | $3^3$, stride=2 | 1 | |
| **LargeKernel3D** | Block 1 | 16 → 32 | $3^3$ | 2 | Xu et al., "LargeKernel3D: Efficient Large Kernels in 3D Conv", CVPR 2023 (CVF Open Access) |
| | Block 2 | 32 → 64 | $3^3$ | 3 | |
| | LargeConv | 64 → 128 | $17^3$ (partitioned) | 1 | |
| | Block 3 | 128 → 128 | $3^3$ | 2 | |
| **VoxelNeXt** | Block 1 | 32 → 64 | $3^3$ | 3 | Zhu et al., "VoxelNeXt: Fully Sparse 3D Detection", CVPR 2023 (CVF Open Access) |
| | Downsample 1 | 64 → 128 | $3^3$, stride=2 | 1 | |
| | Block 2 | 128 → 256 | $3^3$ | 5 | |
| **LinK** | Block 1 | 64 → 64 | $21^3$ (linear) | 3 | Qi et al., "LinK: Linear Kernel Sparse Conv", CVPR 2023 (CVF Open Access) |
| | Downsample 1 | 64 → 128 | $21^3$ (linear) | 1 | |
| | Block 2 | 128 → 256 | $21^3$ (linear) | 3 | |

| Model | Stage | In → Out Channels | Kernel Size | # Blocks | Citation |
|---|---|---|---|---|---|
| **SAFDNet** | Block 1 | 32 → 64 | $3^3$ | 3 | Liu et al., “SAFDNet: Sparse Adaptive Feature Diffusion”, CVPR 2024 (CVF Open Access) |
| | Downsample 1 | 64 → 128 | $3^3$, stride=2 | 1 | |
| | Block 2 | 128 → 256 | $3^3$ | 5 | |