```
import torch
import torch.nn as nn
import numpy as np
import copy
# Not much diff with jit
from numba import jit
"""
TL;DR
-----------------------------------------------------
-----------------------------------------------------
-----------------------------------------------------

Direct sparse algorithm as described in ICLR 17 paper
(implemented using python loops)

50% sparsity --> ~25s
90% sparsity --> ~9s
100% sparsity --> ~38ms
PyTorch conv2d --> ~constant @ 4ms for all sparsity levels

-----------------------------------------------------
-----------------------------------------------------
-----------------------------------------------------
"""
```

```
filter_bank = torch.Tensor([[1, 0, 0], [0, 0, 0], [0, 0, 0]]).view(1, 1, 3, 3).expand(256, 3, -1, -1)
filters = copy.deepcopy(filter_bank)
```

```
feature_map = torch.rand(3, 64, 64)
```

```
filter_bank = filter_bank.contiguous().view(256, -1)
filter_bank.shape
```

```
torch.Size([256, 27])
```

```
feature_map_flat = feature_map.view(-1)
```

```
# For the streaching operation as described in the paper
# Not counting this time in the inference as it is initialization process
def generate_idxs(H, W, C, R, S):
    passed_idx = 0
    idxs = []
    for i in range(C):
        for j in range(R):
            for k in range(S):
                idxs.append(passed_idx)
                passed_idx += 1
            passed_idx = (passed_idx + (W - S))
        passed_idx = (i + 1) * (H * W)
    return idxs
```

```
# Generate 2D sparse weight matrix by streching the weights correctly
def generate_sparse_weight(filter_bank, H, W, C, R, S):
    idxs = generate_idxs(H, W, C, R, S)
    weight = torch.zeros(filter_bank.shape[0], C * H * W)
    for i in range(filter_bank.shape[0]):
        weight[i, idxs] = filter_bank[i, :]
    return weight
```

```
sparse_weight = generate_sparse_weight(filter_bank, 64, 64, 3, 3, 3)
```

```
from scipy.sparse import csr_matrix
```

```
# CSR format in scipy. Torch sparse use COO format.
sparse_csr_weight = csr_matrix(sparse_weight.numpy())
```

```
# Obtain flattened index from c, r, s, H, W
def layout_func_chw(c, r, s, H, W):
    return (c * H + r) * W + s
```

```
# Main algorithm as described in the paper
# I am looping element by element using python for loops which are damn inefficient!
# Need to write C++ extension for this algo
```

```
def sparse_convolution(sparse_csr_weight, feature_map, C, H, W):
    out = np.zeros((sparse_csr_weight.shape[0], H-2, W-2))
    for n in range(sparse_csr_weight.shape[0]):
        for (off, coeff) in zip(sparse_csr_weight[n].indices, sparse_csr_weight[n].data):
            for y in range(0, H - 2):
                for x in range(0, W - 2):
                    out[n, y, x] += coeff * feature_map[off + layout_func_chw(0, y, x, H, W)]
    return out
```

```
%timeit out = sparse_convolution(sparse_csr_weight, feature_map_flat.numpy(), 3, 64, 64)
```

```
1 loop, best of 3: 9.08 s per loop
```

```
feature_map.shape
```

```
torch.Size([3, 64, 64])
```

```
import torch.nn.functional as F
```

```
filter_bank.shape
```

```
torch.Size([256, 27])
```

```
filter_bank = torch.Tensor([[1, 0, 0], [0, 0, 0], [0, 0, 0]]).view(1, 1, 3, 3).expand(256, 3, -1, -1).double()
```

```
# Torch conv2d is ~2K times fast than our pure python looping algo for ~90% sparse kernels
%timeit out_torch = F.conv2d(torch.unsqueeze(feature_map.double(), dim=0), filter_bank)
```

```
The slowest run took 27.72 times longer than the fastest. This could mean that an intermediate result is being cached.
100 loops, best of 3: 4.2 ms per loop
```

```
out = sparse_convolution(sparse_csr_weight, feature_map_flat.numpy(), 3, 64, 64)
out_torch = F.conv2d(torch.unsqueeze(feature_map.double(), dim=0), filter_bank)
```

```
# Just to confirm that algo produce correct convolution output
(out_torch.numpy().squeeze() - out).sum()
```

```
0.0
```

```
# Check speedup for all zeros matrix
filter_bank = torch.Tensor([[0, 0, 0], [0, 0, 0], [0, 0, 0]]).view(1, 1, 3, 3).expand(256, 3, -1, -1)
```

```
sparse_weight = generate_sparse_weight(filter_bank.contiguous().view(256, -1), 64, 64, 3, 3, 3)
sparse_csr_weight = csr_matrix(sparse_weight.numpy())
```

```
# ~230 times speedup when moving from ~90% sparsity to 100% sparsity
# For 50% sparsity though, our algo takes ~25s hence from 50% to 90% we achieve ~2.8 times speedup only
# This indicates that algorithm speeds up when we increase sparsity
%timeit out = sparse_convolution(sparse_csr_weight, feature_map_flat.numpy(), 3, 64, 64)
```

```
10 loops, best of 3: 38.5 ms per loop
```

```
# Pytorch timing remains almost same as it does nothing special to
# consider sparsity
% timeit out_torch = F.conv2d(torch.unsqueeze(feature_map.double(), dim=0), filter_bank.double())
```

```
100 loops, best of 3: 4.46 ms per loop
```

Start coding or generate with AI.