unroll

```
int H_out = H - K + 1;
int W_out = W - K + 1;
for (int b = 0; b < B; ++b)
for (int c = 0; c < C; ++c) {
int w_base = c (KK); for (int p
= 0; p < K; ++p)
for (int q = 0; q < K; ++q) {
for(in th= 0;h< H_out;++h)
for (int w = 0; w < W_out; ++w)
{
int w_unroll = w_base + p * K +
q;
int h_unroll = h * W_out + w;
X_unro ll[b, h_unroll, w_unroll]
= X[b, c, h + p, w + q];
```

replicate

Unroll size CKK x H_out*W_out

Input (/ for rep) C(H_out+K-1)(W_out+K-1)

convo forward

```
int m = blockIdx.x;
int h = blockIdx.y / W_grid +
thread Idx.y;
int w = blockIdx.y % W_grid +
thread Idx.x;
float acc = 0.;
for(in tc=0; c<C ;c++){
for(in tp= 0;p <K;p++) KxK
filter for (int q = 0; q < K;
q++)acc += X[c, h + p, w + q] *
W[m, c, p, q];
}Y[m, h, w] = acc;
```

matrix

```
int X_tile_width = TILE_WIDTH +
K-1;
extern __shared__ float shmem[];
float* X_shared = &s hme m[0];
float W_shared = &s hme m[X -
_ti le_ width X_tile _wi dth];
m = blockI dx.x;
```

matrix (cont)

> h_base = (blockIdx.z / W_grid) TILE_SIZE; the block w_base = (blockIdx.z % W_grid) TILE_SIZE; x = threadIdx.x; ty = thread-Idx.y; h = h_base + tx; w = w_base + ty;
float acc = 0.;
for (c = 0; c < C; c++)
tx and ty used as shorthand for threadIdx.x and threadIdx.y
if ((ty < K) && (tx < K)) W_shared[ty, tx]= W [m, c, ty, tx];
__syncthreads();
for (int i = h; i < h_base + X_tile_width; i += TILE_WIDTH) { for (int j = w; j < w_base + X_tile_width; j += TILE_WIDTH)
X_shared[i - h_base, j - w_base] = X[n, c, i, j]}__syncthreads();}Y[n, m, h, w] = acc;

Histogram

```
__shared__ unsigned int
histo_private[256];
if (threa dIdx.x < 256) histo_ -
pri vat e[t hre adi dx.x] = 0;
__sync thr eads();
int i = thread Idx.x +
blockIdx.x * blockD im.x;
int stride = blockDim.x *
gridDim.x; while (i < size) {
atomicAdd( &( pri vat e_h ist -
o[b uff er[i]), 1);
i += stride; }
__sync thr eads();
if (threa dIdx.x < 256)
atomicAdd( &( his to[ thr ead -
Idx.x]), privat e_h ist o[t -
hre adI dx.x] );
```

CSR ELL COO JDS JDST

```
int row = blockIdx.x *
blockDim.x + threadIdx.x;
if (row < num_rows){
float dot = 0;
int row_start = row_pt r[row];
int row_end = row_pt r[r ow+1];
for (int elem = row_start; elem
< row_end; elem++) {
dot += data[elem] * x[col_ ind -
ex[ elem]]; }
y[row] = dot;
for(in ti= 0;i <nu m_e -
lem ;i++){
dot +=
data[row+inum_rows]x[col_ ind -
ex[ row +i* num _ro ws]]; y[row]
= dot;
for (int i = 0; i < num_elem;
row++)
y[row_ ind ex[i]] += data[i] *
x[col_ ind ex[i]];
int row_start = jds_ro w_p tr[ -
row];
int row_end = jds_ro w_p tr[ -
row+1];
for (int elem = row_start; elem
< row_end; elem++) {
dot += data[elem] * x[col_ ind -
ex[ elem]]; y[jds_ row _in dex -
[row]] = dot;
unsigned in sec = 0;
while (jds_t _co l_p tr[ sec -
+1] -jd s_t _co l_p tr[sec] >
row){ dot += data[j ds_ t_c -
ol_ ptr [se c]+row] *
x[col_ ind ex[ jds _t_ col _pt -
r[s ec] +row]];
      sec++;
y[jds_ row _in dex [row]] = dot;
}
```


Sponsored by Readable.com
Measure your website readability!
https://readable.com