#include "reorg\_layer.h"

#include "dark\_cuda.h"

#include "blas.h"

#include "utils.h"

#include <stdio.h>

layer make\_reorg\_layer(int batch, int w, int h, int c, int stride, int reverse)

{

layer l = { (LAYER\_TYPE)0 };

l.type = REORG;

l.batch = batch;

l.stride = stride;

l.h = h;

l.w = w;

l.c = c;

if(reverse){

l.out\_w = w\*stride;

l.out\_h = h\*stride;

l.out\_c = c/(stride\*stride);

}else{

l.out\_w = w/stride;

l.out\_h = h/stride;

l.out\_c = c\*(stride\*stride);

}

l.reverse = reverse;

fprintf(stderr, "reorg /%2d %4d x%4d x%4d -> %4d x%4d x%4d\n", stride, w, h, c, l.out\_w, l.out\_h, l.out\_c);

l.outputs = l.out\_h \* l.out\_w \* l.out\_c;

l.inputs = h\*w\*c;

int output\_size = l.out\_h \* l.out\_w \* l.out\_c \* batch;

l.output = (float\*)xcalloc(output\_size, sizeof(float));

l.delta = (float\*)xcalloc(output\_size, sizeof(float));

l.forward = forward\_reorg\_layer;

l.backward = backward\_reorg\_layer;

#ifdef GPU

l.forward\_gpu = forward\_reorg\_layer\_gpu;

l.backward\_gpu = backward\_reorg\_layer\_gpu;

l.output\_gpu = cuda\_make\_array(l.output, output\_size);

l.delta\_gpu = cuda\_make\_array(l.delta, output\_size);

#endif

return l;

}

void resize\_reorg\_layer(layer \*l, int w, int h)

{

int stride = l->stride;

int c = l->c;

l->h = h;

l->w = w;

if(l->reverse){

l->out\_w = w\*stride;

l->out\_h = h\*stride;

l->out\_c = c/(stride\*stride);

}else{

l->out\_w = w/stride;

l->out\_h = h/stride;

l->out\_c = c\*(stride\*stride);

}

l->outputs = l->out\_h \* l->out\_w \* l->out\_c;

l->inputs = l->outputs;

int output\_size = l->outputs \* l->batch;

l->output = (float\*)xrealloc(l->output, output\_size \* sizeof(float));

l->delta = (float\*)xrealloc(l->delta, output\_size \* sizeof(float));

#ifdef GPU

cuda\_free(l->output\_gpu);

cuda\_free(l->delta\_gpu);

l->output\_gpu = cuda\_make\_array(l->output, output\_size);

l->delta\_gpu = cuda\_make\_array(l->delta, output\_size);

#endif

}

void forward\_reorg\_layer(const layer l, network\_state state)

{

if (l.reverse) {

reorg\_cpu(state.input, l.out\_w, l.out\_h, l.out\_c, l.batch, l.stride, 1, l.output);

}

else {

reorg\_cpu(state.input, l.out\_w, l.out\_h, l.out\_c, l.batch, l.stride, 0, l.output);

}

}

void backward\_reorg\_layer(const layer l, network\_state state)

{

if (l.reverse) {

reorg\_cpu(l.delta, l.out\_w, l.out\_h, l.out\_c, l.batch, l.stride, 0, state.delta);

}

else {

reorg\_cpu(l.delta, l.out\_w, l.out\_h, l.out\_c, l.batch, l.stride, 1, state.delta);

}

}

#ifdef GPU

void forward\_reorg\_layer\_gpu(layer l, network\_state state)

{

if (l.reverse) {

reorg\_ongpu(state.input, l.out\_w, l.out\_h, l.out\_c, l.batch, l.stride, 1, l.output\_gpu);

}

else {

reorg\_ongpu(state.input, l.out\_w, l.out\_h, l.out\_c, l.batch, l.stride, 0, l.output\_gpu);

}

}

void backward\_reorg\_layer\_gpu(layer l, network\_state state)

{

if (l.reverse) {

reorg\_ongpu(l.delta\_gpu, l.out\_w, l.out\_h, l.out\_c, l.batch, l.stride, 0, state.delta);

}

else {

reorg\_ongpu(l.delta\_gpu, l.out\_w, l.out\_h, l.out\_c, l.batch, l.stride, 1, state.delta);

}

}

#endif