#include "upsample\_layer.h"

#include "dark\_cuda.h"

#include "utils.h"

#include "blas.h"

#include <stdio.h>

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

{

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

l.type = UPSAMPLE;

l.batch = batch;

l.w = w;

l.h = h;

l.c = c;

l.out\_w = w\*stride;

l.out\_h = h\*stride;

l.out\_c = c;

if(stride < 0){

stride = -stride;

l.reverse=1;

l.out\_w = w/stride;

l.out\_h = h/stride;

}

l.stride = stride;

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

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

l.delta = (float\*)xcalloc(l.outputs \* batch, sizeof(float));

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

l.forward = forward\_upsample\_layer;

l.backward = backward\_upsample\_layer;

#ifdef GPU

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

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

l.delta\_gpu = cuda\_make\_array(l.delta, l.outputs\*batch);

l.output\_gpu = cuda\_make\_array(l.output, l.outputs\*batch);

#endif

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

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

return l;

}

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

{

l->w = w;

l->h = h;

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

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

if(l->reverse){

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

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

}

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

l->inputs = l->h\*l->w\*l->c;

l->delta = (float\*)xrealloc(l->delta, l->outputs \* l->batch \* sizeof(float));

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

#ifdef GPU

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

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

l->output\_gpu = cuda\_make\_array(l->output, l->outputs\*l->batch);

l->delta\_gpu = cuda\_make\_array(l->delta, l->outputs\*l->batch);

#endif

}

void forward\_upsample\_layer(const layer l, network\_state net)

{

fill\_cpu(l.outputs\*l.batch, 0, l.output, 1);

if(l.reverse){

upsample\_cpu(l.output, l.out\_w, l.out\_h, l.c, l.batch, l.stride, 0, l.scale, net.input);

}else{

upsample\_cpu(net.input, l.w, l.h, l.c, l.batch, l.stride, 1, l.scale, l.output);

}

}

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

{

if(l.reverse){

upsample\_cpu(l.delta, l.out\_w, l.out\_h, l.c, l.batch, l.stride, 1, l.scale, state.delta);

}else{

upsample\_cpu(state.delta, l.w, l.h, l.c, l.batch, l.stride, 0, l.scale, l.delta);

}

}

#ifdef GPU

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

{

fill\_ongpu(l.outputs\*l.batch, 0, l.output\_gpu, 1);

if(l.reverse){

upsample\_gpu(l.output\_gpu, l.out\_w, l.out\_h, l.c, l.batch, l.stride, 0, l.scale, state.input);

}else{

upsample\_gpu(state.input, l.w, l.h, l.c, l.batch, l.stride, 1, l.scale, l.output\_gpu);

}

}

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

{

if(l.reverse){

upsample\_gpu(l.delta\_gpu, l.out\_w, l.out\_h, l.c, l.batch, l.stride, 1, l.scale, state.delta);

}else{

upsample\_gpu(state.delta, l.w, l.h, l.c, l.batch, l.stride, 0, l.scale, l.delta\_gpu);

}

}

#endif