#include "deconvolutional\_layer.h"

#include "convolutional\_layer.h"

#include "utils.h"

#include "im2col.h"

#include "col2im.h"

#include "blas.h"

#include "gemm.h"

#include <stdio.h>

#include <time.h>

int deconvolutional\_out\_height(deconvolutional\_layer l)

{

int h = l.stride\*(l.h - 1) + l.size;

return h;

}

int deconvolutional\_out\_width(deconvolutional\_layer l)

{

int w = l.stride\*(l.w - 1) + l.size;

return w;

}

int deconvolutional\_out\_size(deconvolutional\_layer l)

{

return deconvolutional\_out\_height(l) \* deconvolutional\_out\_width(l);

}

image get\_deconvolutional\_image(deconvolutional\_layer l)

{

int h,w,c;

h = deconvolutional\_out\_height(l);

w = deconvolutional\_out\_width(l);

c = l.n;

return float\_to\_image(w,h,c,l.output);

}

image get\_deconvolutional\_delta(deconvolutional\_layer l)

{

int h,w,c;

h = deconvolutional\_out\_height(l);

w = deconvolutional\_out\_width(l);

c = l.n;

return float\_to\_image(w,h,c,l.delta);

}

deconvolutional\_layer make\_deconvolutional\_layer(int batch, int h, int w, int c, int n, int size, int stride, ACTIVATION activation)

{

int i;

deconvolutional\_layer l = { (LAYER\_TYPE)0 };

l.type = DECONVOLUTIONAL;

l.h = h;

l.w = w;

l.c = c;

l.n = n;

l.batch = batch;

l.stride = stride;

l.size = size;

l.weights = (float\*)xcalloc(c \* n \* size \* size, sizeof(float));

l.weight\_updates = (float\*)xcalloc(c \* n \* size \* size, sizeof(float));

l.biases = (float\*)xcalloc(n, sizeof(float));

l.bias\_updates = (float\*)xcalloc(n, sizeof(float));

float scale = 1./sqrt(size\*size\*c);

for(i = 0; i < c\*n\*size\*size; ++i) l.weights[i] = scale\*rand\_normal();

for(i = 0; i < n; ++i){

l.biases[i] = scale;

}

int out\_h = deconvolutional\_out\_height(l);

int out\_w = deconvolutional\_out\_width(l);

l.out\_h = out\_h;

l.out\_w = out\_w;

l.out\_c = n;

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

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

l.col\_image = (float\*)xcalloc(h \* w \* size \* size \* n, sizeof(float));

l.output = (float\*)xcalloc(l.batch \* out\_h \* out\_w \* n, sizeof(float));

l.delta = (float\*)xcalloc(l.batch \* out\_h \* out\_w \* n, sizeof(float));

l.forward = forward\_deconvolutional\_layer;

l.backward = backward\_deconvolutional\_layer;

l.update = update\_deconvolutional\_layer;

#ifdef GPU

l.weights\_gpu = cuda\_make\_array(l.weights, c\*n\*size\*size);

l.weight\_updates\_gpu = cuda\_make\_array(l.weight\_updates, c\*n\*size\*size);

l.biases\_gpu = cuda\_make\_array(l.biases, n);

l.bias\_updates\_gpu = cuda\_make\_array(l.bias\_updates, n);

l.col\_image\_gpu = cuda\_make\_array(l.col\_image, h\*w\*size\*size\*n);

l.delta\_gpu = cuda\_make\_array(l.delta, l.batch\*out\_h\*out\_w\*n);

l.output\_gpu = cuda\_make\_array(l.output, l.batch\*out\_h\*out\_w\*n);

#endif

l.activation = activation;

fprintf(stderr, "Deconvolutional Layer: %d x %d x %d image, %d filters -> %d x %d x %d image\n", h,w,c,n, out\_h, out\_w, n);

return l;

}

void resize\_deconvolutional\_layer(deconvolutional\_layer \*l, int h, int w)

{

l->h = h;

l->w = w;

int out\_h = deconvolutional\_out\_height(\*l);

int out\_w = deconvolutional\_out\_width(\*l);

l->col\_image = (float\*)xrealloc(l->col\_image,

out\_h\*out\_w\*l->size\*l->size\*l->c\*sizeof(float));

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

l->batch\*out\_h \* out\_w \* l->n\*sizeof(float));

l->delta = (float\*)xrealloc(l->delta,

l->batch\*out\_h \* out\_w \* l->n\*sizeof(float));

#ifdef GPU

cuda\_free(l->col\_image\_gpu);

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

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

l->col\_image\_gpu = cuda\_make\_array(l->col\_image, out\_h\*out\_w\*l->size\*l->size\*l->c);

l->delta\_gpu = cuda\_make\_array(l->delta, l->batch\*out\_h\*out\_w\*l->n);

l->output\_gpu = cuda\_make\_array(l->output, l->batch\*out\_h\*out\_w\*l->n);

#endif

}

void forward\_deconvolutional\_layer(const deconvolutional\_layer l, network\_state state)

{

int i;

int out\_h = deconvolutional\_out\_height(l);

int out\_w = deconvolutional\_out\_width(l);

int size = out\_h\*out\_w;

int m = l.size\*l.size\*l.n;

int n = l.h\*l.w;

int k = l.c;

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

for(i = 0; i < l.batch; ++i){

float \*a = l.weights;

float \*b = state.input + i\*l.c\*l.h\*l.w;

float \*c = l.col\_image;

gemm(1,0,m,n,k,1,a,m,b,n,0,c,n);

col2im\_cpu(c, l.n, out\_h, out\_w, l.size, l.stride, 0, l.output+i\*l.n\*size);

}

add\_bias(l.output, l.biases, l.batch, l.n, size);

activate\_array(l.output, l.batch\*l.n\*size, l.activation);

}

void backward\_deconvolutional\_layer(deconvolutional\_layer l, network\_state state)

{

float alpha = 1./l.batch;

int out\_h = deconvolutional\_out\_height(l);

int out\_w = deconvolutional\_out\_width(l);

int size = out\_h\*out\_w;

int i;

gradient\_array(l.output, size\*l.n\*l.batch, l.activation, l.delta);

backward\_bias(l.bias\_updates, l.delta, l.batch, l.n, size);

for(i = 0; i < l.batch; ++i){

int m = l.c;

int n = l.size\*l.size\*l.n;

int k = l.h\*l.w;

float \*a = state.input + i\*m\*n;

float \*b = l.col\_image;

float \*c = l.weight\_updates;

im2col\_cpu(l.delta + i\*l.n\*size, l.n, out\_h, out\_w,

l.size, l.stride, 0, b);

gemm(0,1,m,n,k,alpha,a,k,b,k,1,c,n);

if(state.delta){

int m = l.c;

int n = l.h\*l.w;

int k = l.size\*l.size\*l.n;

float \*a = l.weights;

float \*b = l.col\_image;

float \*c = state.delta + i\*n\*m;

gemm(0,0,m,n,k,1,a,k,b,n,1,c,n);

}

}

}

void update\_deconvolutional\_layer(deconvolutional\_layer l, int skip, float learning\_rate, float momentum, float decay)

{

int size = l.size\*l.size\*l.c\*l.n;

axpy\_cpu(l.n, learning\_rate, l.bias\_updates, 1, l.biases, 1);

scal\_cpu(l.n, momentum, l.bias\_updates, 1);

axpy\_cpu(size, -decay, l.weights, 1, l.weight\_updates, 1);

axpy\_cpu(size, learning\_rate, l.weight\_updates, 1, l.weights, 1);

scal\_cpu(size, momentum, l.weight\_updates, 1);

}