#ifndef CONVOLUTIONAL\_LAYER\_H

#define CONVOLUTIONAL\_LAYER\_H

#include "dark\_cuda.h"

#include "image.h"

#include "activations.h"

#include "layer.h"

#include "network.h"

typedef layer convolutional\_layer;

#ifdef \_\_cplusplus

extern "C" {

#endif

#ifdef GPU

void forward\_convolutional\_layer\_gpu(convolutional\_layer layer, network\_state state);

void backward\_convolutional\_layer\_gpu(convolutional\_layer layer, network\_state state);

void update\_convolutional\_layer\_gpu(convolutional\_layer layer, int batch, float learning\_rate, float momentum, float decay, float loss\_scale);

void push\_convolutional\_layer(convolutional\_layer layer);

void pull\_convolutional\_layer(convolutional\_layer layer);

void add\_bias\_gpu(float \*output, float \*biases, int batch, int n, int size);

void backward\_bias\_gpu(float \*bias\_updates, float \*delta, int batch, int n, int size);

#ifdef CUDNN

void cudnn\_convolutional\_setup(layer \*l, int cudnn\_preference, size\_t workspace\_size\_specify);

void create\_convolutional\_cudnn\_tensors(layer \*l);

void cuda\_convert\_f32\_to\_f16(float\* input\_f32, size\_t size, float \*output\_f16);

#endif

#endif

void free\_convolutional\_batchnorm(convolutional\_layer \*l);

size\_t get\_convolutional\_workspace\_size(layer l);

convolutional\_layer make\_convolutional\_layer(int batch, int steps, int h, int w, int c, int n, int groups, int size, int stride\_x, int stride\_y, int dilation, int padding, ACTIVATION activation, int batch\_normalize, int binary, int xnor, int adam, int use\_bin\_output, int index, int antialiasing, convolutional\_layer \*share\_layer, int assisted\_excitation, int deform, int train);

void denormalize\_convolutional\_layer(convolutional\_layer l);

void set\_specified\_workspace\_limit(convolutional\_layer \*l, size\_t workspace\_size\_limit);

void resize\_convolutional\_layer(convolutional\_layer \*layer, int w, int h);

void forward\_convolutional\_layer(const convolutional\_layer layer, network\_state state);

void update\_convolutional\_layer(convolutional\_layer layer, int batch, float learning\_rate, float momentum, float decay);

image \*visualize\_convolutional\_layer(convolutional\_layer layer, char \*window, image \*prev\_weights);

void binarize\_weights(float \*weights, int n, int size, float \*binary);

void swap\_binary(convolutional\_layer \*l);

void binarize\_weights2(float \*weights, int n, int size, char \*binary, float \*scales);

void binary\_align\_weights(convolutional\_layer \*l);

void backward\_convolutional\_layer(convolutional\_layer layer, network\_state state);

void add\_bias(float \*output, float \*biases, int batch, int n, int size);

void backward\_bias(float \*bias\_updates, float \*delta, int batch, int n, int size);

image get\_convolutional\_image(convolutional\_layer layer);

image get\_convolutional\_delta(convolutional\_layer layer);

image get\_convolutional\_weight(convolutional\_layer layer, int i);

int convolutional\_out\_height(convolutional\_layer layer);

int convolutional\_out\_width(convolutional\_layer layer);

void rescale\_weights(convolutional\_layer l, float scale, float trans);

void rgbgr\_weights(convolutional\_layer l);

void assisted\_excitation\_forward(convolutional\_layer l, network\_state state);

void assisted\_excitation\_forward\_gpu(convolutional\_layer l, network\_state state);

#ifdef \_\_cplusplus

}

#endif

#endif