#ifndef BLAS\_H

#define BLAS\_H

#include <stdlib.h>

#include "darknet.h"

#ifdef GPU

#include "dark\_cuda.h"

#include "tree.h"

#endif

#ifdef \_\_cplusplus

extern "C" {

#endif

void flatten(float \*x, int size, int layers, int batch, int forward);

void pm(int M, int N, float \*A);

float \*random\_matrix(int rows, int cols);

void time\_random\_matrix(int TA, int TB, int m, int k, int n);

void reorg\_cpu(float \*x, int w, int h, int c, int batch, int stride, int forward, float \*out);

void test\_blas();

void const\_cpu(int N, float ALPHA, float \*X, int INCX);

void constrain\_ongpu(int N, float ALPHA, float \* X, int INCX);

void constrain\_min\_max\_ongpu(int N, float MIN, float MAX, float \* X, int INCX);

void pow\_cpu(int N, float ALPHA, float \*X, int INCX, float \*Y, int INCY);

void mul\_cpu(int N, float \*X, int INCX, float \*Y, int INCY);

void axpy\_cpu(int N, float ALPHA, float \*X, int INCX, float \*Y, int INCY);

void copy\_cpu(int N, float \*X, int INCX, float \*Y, int INCY);

void scal\_cpu(int N, float ALPHA, float \*X, int INCX);

void scal\_add\_cpu(int N, float ALPHA, float BETA, float \*X, int INCX);

void fill\_cpu(int N, float ALPHA, float \* X, int INCX);

float dot\_cpu(int N, float \*X, int INCX, float \*Y, int INCY);

void test\_gpu\_blas();

void shortcut\_cpu(int batch, int w1, int h1, int c1, float \*add, int w2, int h2, int c2, float \*out);

void shortcut\_multilayer\_cpu(int size, int src\_outputs, int batch, int n, int \*outputs\_of\_layers, float \*\*layers\_output, float \*out, float \*in, float \*weights, int nweights, WEIGHTS\_NORMALIZATION\_T weights\_normalizion);

void backward\_shortcut\_multilayer\_cpu(int size, int src\_outputs, int batch, int n, int \*outputs\_of\_layers,

float \*\*layers\_delta, float \*delta\_out, float \*delta\_in, float \*weights, float \*weight\_updates, int nweights, float \*in, float \*\*layers\_output, WEIGHTS\_NORMALIZATION\_T weights\_normalizion);

void mean\_cpu(float \*x, int batch, int filters, int spatial, float \*mean);

void variance\_cpu(float \*x, float \*mean, int batch, int filters, int spatial, float \*variance);

void normalize\_cpu(float \*x, float \*mean, float \*variance, int batch, int filters, int spatial);

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

void scale\_bias(float \*output, float \*scales, int batch, int n, int size);

void backward\_scale\_cpu(float \*x\_norm, float \*delta, int batch, int n, int size, float \*scale\_updates);

void mean\_delta\_cpu(float \*delta, float \*variance, int batch, int filters, int spatial, float \*mean\_delta);

void variance\_delta\_cpu(float \*x, float \*delta, float \*mean, float \*variance, int batch, int filters, int spatial, float \*variance\_delta);

void normalize\_delta\_cpu(float \*x, float \*mean, float \*variance, float \*mean\_delta, float \*variance\_delta, int batch, int filters, int spatial, float \*delta);

void smooth\_l1\_cpu(int n, float \*pred, float \*truth, float \*delta, float \*error);

void l2\_cpu(int n, float \*pred, float \*truth, float \*delta, float \*error);

void weighted\_sum\_cpu(float \*a, float \*b, float \*s, int num, float \*c);

void softmax(float \*input, int n, float temp, float \*output, int stride);

void upsample\_cpu(float \*in, int w, int h, int c, int batch, int stride, int forward, float scale, float \*out);

void softmax\_cpu(float \*input, int n, int batch, int batch\_offset, int groups, int group\_offset, int stride, float temp, float \*output);

void softmax\_x\_ent\_cpu(int n, float \*pred, float \*truth, float \*delta, float \*error);

void constrain\_cpu(int size, float ALPHA, float \*X);

void fix\_nan\_and\_inf\_cpu(float \*input, size\_t size);

#ifdef GPU

void constrain\_weight\_updates\_ongpu(int N, float coef, float \*weights\_gpu, float \*weight\_updates\_gpu);

void axpy\_ongpu(int N, float ALPHA, float \* X, int INCX, float \* Y, int INCY);

void axpy\_ongpu\_offset(int N, float ALPHA, float \* X, int OFFX, int INCX, float \* Y, int OFFY, int INCY);

void simple\_copy\_ongpu(int size, float \*src, float \*dst);

void memcpy\_ongpu(void \*dst, void \*src, int size\_bytes);

void copy\_ongpu(int N, float \* X, int INCX, float \* Y, int INCY);

void copy\_ongpu\_offset(int N, float \* X, int OFFX, int INCX, float \* Y, int OFFY, int INCY);

void scal\_ongpu(int N, float ALPHA, float \* X, int INCX);

void scal\_add\_ongpu(int N, float ALPHA, float BETA, float \* X, int INCX);

void supp\_ongpu(int N, float ALPHA, float \* X, int INCX);

void mask\_gpu\_new\_api(int N, float \* X, float mask\_num, float \* mask, float val);

void mask\_ongpu(int N, float \* X, float mask\_num, float \* mask);

void const\_ongpu(int N, float ALPHA, float \*X, int INCX);

void pow\_ongpu(int N, float ALPHA, float \*X, int INCX, float \*Y, int INCY);

void mul\_ongpu(int N, float \*X, int INCX, float \*Y, int INCY);

void fill\_ongpu(int N, float ALPHA, float \* X, int INCX);

void mean\_gpu(float \*x, int batch, int filters, int spatial, float \*mean);

void variance\_gpu(float \*x, float \*mean, int batch, int filters, int spatial, float \*variance);

void normalize\_gpu(float \*x, float \*mean, float \*variance, int batch, int filters, int spatial);

void normalize\_delta\_gpu(float \*x, float \*mean, float \*variance, float \*mean\_delta, float \*variance\_delta, int batch, int filters, int spatial, float \*delta);

void fast\_mean\_delta\_gpu(float \*delta, float \*variance, int batch, int filters, int spatial, float \*mean\_delta);

void fast\_variance\_delta\_gpu(float \*x, float \*delta, float \*mean, float \*variance, int batch, int filters, int spatial, float \*variance\_delta);

void fast\_mean\_gpu(float \*x, int batch, int filters, int spatial, float \*mean);

void fast\_variance\_gpu(float \*x, float \*mean, int batch, int filters, int spatial, float \*variance);

void fast\_v\_cbn\_gpu(const float \*x, float \*mean, int batch, int filters, int spatial, int minibatch\_index, int max\_minibatch\_index, float \*m\_avg, float \*v\_avg, float \*variance,

const float alpha, float \*rolling\_mean\_gpu, float \*rolling\_variance\_gpu, int inverse\_variance, float epsilon);

void inverse\_variance\_ongpu(int size, float \*src, float \*dst, float epsilon);

void normalize\_scale\_bias\_gpu(float \*x, float \*mean, float \*variance, float \*scales, float \*biases, int batch, int filters, int spatial, int inverse\_variance, float epsilon);

void compare\_2\_arrays\_gpu(float \*one, float \*two, int size);

void shortcut\_gpu(int batch, int w1, int h1, int c1, float \*add, int w2, int h2, int c2, float \*out);

void shortcut\_multilayer\_gpu(int src\_outputs, int batch, int n, int \*outputs\_of\_layers\_gpu, float \*\*layers\_output\_gpu, float \*out, float \*in, float \*weights\_gpu, int nweights, WEIGHTS\_NORMALIZATION\_T weights\_normalizion);

void backward\_shortcut\_multilayer\_gpu(int src\_outputs, int batch, int n, int \*outputs\_of\_layers\_gpu, float \*\*layers\_delta\_gpu, float \*delta\_out, float \*delta\_in,

float \*weights, float \*weight\_updates, int nweights, float \*in, float \*\*layers\_output, WEIGHTS\_NORMALIZATION\_T weights\_normalizion);

void input\_shortcut\_gpu(float \*in, int batch, int w1, int h1, int c1, float \*add, int w2, int h2, int c2, float \*out);

void backward\_scale\_gpu(float \*x\_norm, float \*delta, int batch, int n, int size, float \*scale\_updates);

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

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);

void softmax\_x\_ent\_gpu(int n, float \*pred, float \*truth, float \*delta, float \*error);

void smooth\_l1\_gpu(int n, float \*pred, float \*truth, float \*delta, float \*error);

void l2\_gpu(int n, float \*pred, float \*truth, float \*delta, float \*error);

void weighted\_delta\_gpu(float \*a, float \*b, float \*s, float \*da, float \*db, float \*ds, int num, float \*dc);

void weighted\_sum\_gpu(float \*a, float \*b, float \*s, int num, float \*c);

void mult\_add\_into\_gpu(int num, float \*a, float \*b, float \*c);

void reorg\_ongpu(float \*x, int w, int h, int c, int batch, int stride, int forward, float \*out);

void softmax\_gpu\_new\_api(float \*input, int n, int batch, int batch\_offset, int groups, int group\_offset, int stride, float temp, float \*output);

void softmax\_gpu(float \*input, int n, int offset, int groups, float temp, float \*output);

void adam\_gpu(int n, float \*x, float \*m, float \*v, float B1, float B2, float rate, float eps, int t);

void adam\_update\_gpu(float \*w, float \*d, float \*m, float \*v, float B1, float B2, float eps, float decay, float rate, int n, int batch, int t);

void flatten\_ongpu(float \*x, int spatial, int layers, int batch, int forward, float \*out);

void upsample\_gpu(float \*in, int w, int h, int c, int batch, int stride, int forward, float scale, float \*out);

void softmax\_tree\_gpu(float \*input, int spatial, int batch, int stride, float temp, float \*output, tree hier);

void fix\_nan\_and\_inf(float \*input, size\_t size);

void reset\_nan\_and\_inf(float \*input, size\_t size);

int is\_nan\_or\_inf(float \*input, size\_t size);

void add\_3\_arrays\_activate(float \*a1, float \*a2, float \*a3, size\_t size, ACTIVATION a, float \*dst);

void sum\_of\_mults(float \*a1, float \*a2, float \*b1, float \*b2, size\_t size, float \*dst);

void activate\_and\_mult(float \*a1, float \*a2, size\_t size, ACTIVATION a, float \*dst);

void scale\_channels\_gpu(float \*in\_w\_h\_c, int size, int channel\_size, int batch\_size, int scale\_wh, float \*scales\_c, float \*out);

void backward\_scale\_channels\_gpu(float \*in\_w\_h\_c\_delta, int size, int channel\_size, int batch\_size, int scale\_wh,

float \*in\_scales\_c, float \*out\_from\_delta,

float \*in\_from\_output, float \*out\_state\_delta);

void backward\_sam\_gpu(float \*in\_w\_h\_c\_delta, int size, int channel\_size,

float \*in\_scales\_c, float \*out\_from\_delta,

float \*in\_from\_output, float \*out\_state\_delta);

void sam\_gpu(float \*in\_w\_h\_c, int size, int channel\_size, float \*scales\_c, float \*out);

void smooth\_rotate\_weights\_gpu(const float \*src\_weight\_gpu, float \*weight\_deform\_gpu, int nweights, int n, int size, int angle, int reverse);

void stretch\_weights\_gpu(const float \*src\_weight\_gpu, float \*weight\_deform\_gpu, int nweights, int n, int size, float scale, int reverse);

void sway\_and\_flip\_weights\_gpu(const float \*src\_weight\_gpu, float \*weight\_deform\_gpu, int nweights, int n, int size, int angle, int reverse);

void stretch\_sway\_flip\_weights\_gpu(const float \*src\_weight\_gpu, float \*weight\_deform\_gpu, int nweights, int n, int size, int angle, int reverse);

void rotate\_weights\_gpu(const float \*src\_weight\_gpu, float \*weight\_deform\_gpu, int nweights, int n, int size, int reverse);

void reduce\_and\_expand\_array\_gpu(const float \*src\_gpu, float \*dst\_gpu, int size, int groups);

void expand\_array\_gpu(const float \*src\_gpu, float \*dst\_gpu, int size, int groups);

#endif

#ifdef \_\_cplusplus

}

#endif

#endif