#ifndef DARKNET\_API

#define DARKNET\_API

#if defined(\_MSC\_VER) && \_MSC\_VER < 1900

#define inline \_\_inline

#endif

#if defined(DEBUG) && !defined(\_CRTDBG\_MAP\_ALLOC)

#define \_CRTDBG\_MAP\_ALLOC

#endif

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <stdint.h>

#include <assert.h>

#include <pthread.h>

#ifndef LIB\_API

#ifdef LIB\_EXPORTS

#if defined(\_MSC\_VER)

#define LIB\_API \_\_declspec(dllexport)

#else

#define LIB\_API \_\_attribute\_\_((visibility("default")))

#endif

#else

#if defined(\_MSC\_VER)

#define LIB\_API

#else

#define LIB\_API

#endif

#endif

#endif

#define SECRET\_NUM -1234

typedef enum { UNUSED\_DEF\_VAL } UNUSED\_ENUM\_TYPE;

#ifdef GPU

#include <cuda\_runtime.h>

#include <curand.h>

#include <cublas\_v2.h>

#ifdef CUDNN

#include <cudnn.h>

#endif // CUDNN

#endif // GPU

#ifdef \_\_cplusplus

extern "C" {

#endif

struct network;

typedef struct network network;

struct network\_state;

typedef struct network\_state network\_state;

struct layer;

typedef struct layer layer;

struct image;

typedef struct image image;

struct detection;

typedef struct detection detection;

struct load\_args;

typedef struct load\_args load\_args;

struct data;

typedef struct data data;

struct metadata;

typedef struct metadata metadata;

struct tree;

typedef struct tree tree;

extern int gpu\_index;

// option\_list.h

typedef struct metadata {

int classes;

char \*\*names;

} metadata;

// tree.h

typedef struct tree {

int \*leaf;

int n;

int \*parent;

int \*child;

int \*group;

char \*\*name;

int groups;

int \*group\_size;

int \*group\_offset;

} tree;

// activations.h

typedef enum {

LOGISTIC, RELU, RELU6, RELIE, LINEAR, RAMP, TANH, PLSE, LEAKY, ELU, LOGGY, STAIR, HARDTAN, LHTAN, SELU, GELU, SWISH, MISH, NORM\_CHAN, NORM\_CHAN\_SOFTMAX, NORM\_CHAN\_SOFTMAX\_MAXVAL

}ACTIVATION;

// parser.h

typedef enum {

IOU, GIOU, MSE, DIOU, CIOU

} IOU\_LOSS;

// parser.h

typedef enum {

DEFAULT\_NMS, GREEDY\_NMS, DIOU\_NMS, CORNERS\_NMS

} NMS\_KIND;

// parser.h

typedef enum {

YOLO\_CENTER = 1 << 0, YOLO\_LEFT\_TOP = 1 << 1, YOLO\_RIGHT\_BOTTOM = 1 << 2

} YOLO\_POINT;

// parser.h

typedef enum {

NO\_WEIGHTS, PER\_FEATURE, PER\_CHANNEL

} WEIGHTS\_TYPE\_T;

// parser.h

typedef enum {

NO\_NORMALIZATION, RELU\_NORMALIZATION, SOFTMAX\_NORMALIZATION

} WEIGHTS\_NORMALIZATION\_T;

// image.h

typedef enum{

PNG, BMP, TGA, JPG

} IMTYPE;

// activations.h

typedef enum{

MULT, ADD, SUB, DIV

} BINARY\_ACTIVATION;

// layer.h

typedef enum {

CONVOLUTIONAL,

DECONVOLUTIONAL,

CONNECTED,

MAXPOOL,

LOCAL\_AVGPOOL,

SOFTMAX,

DETECTION,

DROPOUT,

CROP,

ROUTE,

COST,

NORMALIZATION,

AVGPOOL,

LOCAL,

SHORTCUT,

SCALE\_CHANNELS,

SAM,

ACTIVE,

RNN,

GRU,

LSTM,

CONV\_LSTM,

CRNN,

BATCHNORM,

NETWORK,

XNOR,

REGION,

YOLO,

GAUSSIAN\_YOLO,

ISEG,

REORG,

REORG\_OLD,

UPSAMPLE,

LOGXENT,

L2NORM,

EMPTY,

BLANK

} LAYER\_TYPE;

// layer.h

typedef enum{

SSE, MASKED, L1, SEG, SMOOTH,WGAN

} COST\_TYPE;

// layer.h

typedef struct update\_args {

int batch;

float learning\_rate;

float momentum;

float decay;

int adam;

float B1;

float B2;

float eps;

int t;

} update\_args;

// layer.h

struct layer {

LAYER\_TYPE type;

ACTIVATION activation;

COST\_TYPE cost\_type;

void(\*forward) (struct layer, struct network\_state);

void(\*backward) (struct layer, struct network\_state);

void(\*update) (struct layer, int, float, float, float);

void(\*forward\_gpu) (struct layer, struct network\_state);

void(\*backward\_gpu) (struct layer, struct network\_state);

void(\*update\_gpu) (struct layer, int, float, float, float, float);

layer \*share\_layer;

int train;

int avgpool;

int batch\_normalize;

int shortcut;

int batch;

int dynamic\_minibatch;

int forced;

int flipped;

int inputs;

int outputs;

int nweights;

int nbiases;

int extra;

int truths;

int h, w, c;

int out\_h, out\_w, out\_c;

int n;

int max\_boxes;

int groups;

int group\_id;

int size;

int side;

int stride;

int stride\_x;

int stride\_y;

int dilation;

int antialiasing;

int maxpool\_depth;

int out\_channels;

int reverse;

int flatten;

int spatial;

int pad;

int sqrt;

int flip;

int index;

int scale\_wh;

int binary;

int xnor;

int peephole;

int use\_bin\_output;

int keep\_delta\_gpu;

int optimized\_memory;

int steps;

int state\_constrain;

int hidden;

int truth;

float smooth;

float dot;

int deform;

int sway;

int rotate;

int stretch;

int stretch\_sway;

float angle;

float jitter;

float saturation;

float exposure;

float shift;

float ratio;

float learning\_rate\_scale;

float clip;

int focal\_loss;

float \*classes\_multipliers;

float label\_smooth\_eps;

int noloss;

int softmax;

int classes;

int coords;

int background;

int rescore;

int objectness;

int does\_cost;

int joint;

int noadjust;

int reorg;

int log;

int tanh;

int \*mask;

int total;

float bflops;

int adam;

float B1;

float B2;

float eps;

int t;

float alpha;

float beta;

float kappa;

float coord\_scale;

float object\_scale;

float noobject\_scale;

float mask\_scale;

float class\_scale;

int bias\_match;

float random;

float ignore\_thresh;

float truth\_thresh;

float iou\_thresh;

float thresh;

float focus;

int classfix;

int absolute;

int assisted\_excitation;

int onlyforward;

int stopbackward;

int train\_only\_bn;

int dont\_update;

int burnin\_update;

int dontload;

int dontsave;

int dontloadscales;

int numload;

float temperature;

float probability;

float dropblock\_size\_rel;

int dropblock\_size\_abs;

int dropblock;

float scale;

int receptive\_w;

int receptive\_h;

int receptive\_w\_scale;

int receptive\_h\_scale;

char \* cweights;

int \* indexes;

int \* input\_layers;

int \* input\_sizes;

float \*\*layers\_output;

float \*\*layers\_delta;

WEIGHTS\_TYPE\_T weights\_type;

WEIGHTS\_NORMALIZATION\_T weights\_normalizion;

int \* map;

int \* counts;

float \*\* sums;

float \* rand;

float \* cost;

float \* state;

float \* prev\_state;

float \* forgot\_state;

float \* forgot\_delta;

float \* state\_delta;

float \* combine\_cpu;

float \* combine\_delta\_cpu;

float \*concat;

float \*concat\_delta;

float \*binary\_weights;

float \*biases;

float \*bias\_updates;

float \*scales;

float \*scale\_updates;

float \*weights;

float \*weight\_updates;

float scale\_x\_y;

float max\_delta;

float uc\_normalizer;

float iou\_normalizer;

float cls\_normalizer;

IOU\_LOSS iou\_loss;

IOU\_LOSS iou\_thresh\_kind;

NMS\_KIND nms\_kind;

float beta\_nms;

YOLO\_POINT yolo\_point;

char \*align\_bit\_weights\_gpu;

float \*mean\_arr\_gpu;

float \*align\_workspace\_gpu;

float \*transposed\_align\_workspace\_gpu;

int align\_workspace\_size;

char \*align\_bit\_weights;

float \*mean\_arr;

int align\_bit\_weights\_size;

int lda\_align;

int new\_lda;

int bit\_align;

float \*col\_image;

float \* delta;

float \* output;

float \* activation\_input;

int delta\_pinned;

int output\_pinned;

float \* loss;

float \* squared;

float \* norms;

float \* spatial\_mean;

float \* mean;

float \* variance;

float \* mean\_delta;

float \* variance\_delta;

float \* rolling\_mean;

float \* rolling\_variance;

float \* x;

float \* x\_norm;

float \* m;

float \* v;

float \* bias\_m;

float \* bias\_v;

float \* scale\_m;

float \* scale\_v;

float \*z\_cpu;

float \*r\_cpu;

float \*h\_cpu;

float \*stored\_h\_cpu;

float \* prev\_state\_cpu;

float \*temp\_cpu;

float \*temp2\_cpu;

float \*temp3\_cpu;

float \*dh\_cpu;

float \*hh\_cpu;

float \*prev\_cell\_cpu;

float \*cell\_cpu;

float \*f\_cpu;

float \*i\_cpu;

float \*g\_cpu;

float \*o\_cpu;

float \*c\_cpu;

float \*stored\_c\_cpu;

float \*dc\_cpu;

float \*binary\_input;

uint32\_t \*bin\_re\_packed\_input;

char \*t\_bit\_input;

struct layer \*input\_layer;

struct layer \*self\_layer;

struct layer \*output\_layer;

struct layer \*reset\_layer;

struct layer \*update\_layer;

struct layer \*state\_layer;

struct layer \*input\_gate\_layer;

struct layer \*state\_gate\_layer;

struct layer \*input\_save\_layer;

struct layer \*state\_save\_layer;

struct layer \*input\_state\_layer;

struct layer \*state\_state\_layer;

struct layer \*input\_z\_layer;

struct layer \*state\_z\_layer;

struct layer \*input\_r\_layer;

struct layer \*state\_r\_layer;

struct layer \*input\_h\_layer;

struct layer \*state\_h\_layer;

struct layer \*wz;

struct layer \*uz;

struct layer \*wr;

struct layer \*ur;

struct layer \*wh;

struct layer \*uh;

struct layer \*uo;

struct layer \*wo;

struct layer \*vo;

struct layer \*uf;

struct layer \*wf;

struct layer \*vf;

struct layer \*ui;

struct layer \*wi;

struct layer \*vi;

struct layer \*ug;

struct layer \*wg;

tree \*softmax\_tree;

size\_t workspace\_size;

//#ifdef GPU

int \*indexes\_gpu;

float \*z\_gpu;

float \*r\_gpu;

float \*h\_gpu;

float \*stored\_h\_gpu;

float \*temp\_gpu;

float \*temp2\_gpu;

float \*temp3\_gpu;

float \*dh\_gpu;

float \*hh\_gpu;

float \*prev\_cell\_gpu;

float \*prev\_state\_gpu;

float \*last\_prev\_state\_gpu;

float \*last\_prev\_cell\_gpu;

float \*cell\_gpu;

float \*f\_gpu;

float \*i\_gpu;

float \*g\_gpu;

float \*o\_gpu;

float \*c\_gpu;

float \*stored\_c\_gpu;

float \*dc\_gpu;

// adam

float \*m\_gpu;

float \*v\_gpu;

float \*bias\_m\_gpu;

float \*scale\_m\_gpu;

float \*bias\_v\_gpu;

float \*scale\_v\_gpu;

float \* combine\_gpu;

float \* combine\_delta\_gpu;

float \* forgot\_state\_gpu;

float \* forgot\_delta\_gpu;

float \* state\_gpu;

float \* state\_delta\_gpu;

float \* gate\_gpu;

float \* gate\_delta\_gpu;

float \* save\_gpu;

float \* save\_delta\_gpu;

float \* concat\_gpu;

float \* concat\_delta\_gpu;

float \*binary\_input\_gpu;

float \*binary\_weights\_gpu;

float \*bin\_conv\_shortcut\_in\_gpu;

float \*bin\_conv\_shortcut\_out\_gpu;

float \* mean\_gpu;

float \* variance\_gpu;

float \* m\_cbn\_avg\_gpu;

float \* v\_cbn\_avg\_gpu;

float \* rolling\_mean\_gpu;

float \* rolling\_variance\_gpu;

float \* variance\_delta\_gpu;

float \* mean\_delta\_gpu;

float \* col\_image\_gpu;

float \* x\_gpu;

float \* x\_norm\_gpu;

float \* weights\_gpu;

float \* weight\_updates\_gpu;

float \* weight\_deform\_gpu;

float \* weight\_change\_gpu;

float \* weights\_gpu16;

float \* weight\_updates\_gpu16;

float \* biases\_gpu;

float \* bias\_updates\_gpu;

float \* bias\_change\_gpu;

float \* scales\_gpu;

float \* scale\_updates\_gpu;

float \* scale\_change\_gpu;

float \* input\_antialiasing\_gpu;

float \* output\_gpu;

float \* activation\_input\_gpu;

float \* loss\_gpu;

float \* delta\_gpu;

float \* rand\_gpu;

float \* drop\_blocks\_scale;

float \* drop\_blocks\_scale\_gpu;

float \* squared\_gpu;

float \* norms\_gpu;

float \*gt\_gpu;

float \*a\_avg\_gpu;

int \*input\_sizes\_gpu;

float \*\*layers\_output\_gpu;

float \*\*layers\_delta\_gpu;

#ifdef CUDNN

cudnnTensorDescriptor\_t srcTensorDesc, dstTensorDesc;

cudnnTensorDescriptor\_t srcTensorDesc16, dstTensorDesc16;

cudnnTensorDescriptor\_t dsrcTensorDesc, ddstTensorDesc;

cudnnTensorDescriptor\_t dsrcTensorDesc16, ddstTensorDesc16;

cudnnTensorDescriptor\_t normTensorDesc, normDstTensorDesc, normDstTensorDescF16;

cudnnFilterDescriptor\_t weightDesc, weightDesc16;

cudnnFilterDescriptor\_t dweightDesc, dweightDesc16;

cudnnConvolutionDescriptor\_t convDesc;

cudnnConvolutionFwdAlgo\_t fw\_algo, fw\_algo16;

cudnnConvolutionBwdDataAlgo\_t bd\_algo, bd\_algo16;

cudnnConvolutionBwdFilterAlgo\_t bf\_algo, bf\_algo16;

cudnnPoolingDescriptor\_t poolingDesc;

#else // CUDNN

void\* srcTensorDesc, \*dstTensorDesc;

void\* srcTensorDesc16, \*dstTensorDesc16;

void\* dsrcTensorDesc, \*ddstTensorDesc;

void\* dsrcTensorDesc16, \*ddstTensorDesc16;

void\* normTensorDesc, \*normDstTensorDesc, \*normDstTensorDescF16;

void\* weightDesc, \*weightDesc16;

void\* dweightDesc, \*dweightDesc16;

void\* convDesc;

UNUSED\_ENUM\_TYPE fw\_algo, fw\_algo16;

UNUSED\_ENUM\_TYPE bd\_algo, bd\_algo16;

UNUSED\_ENUM\_TYPE bf\_algo, bf\_algo16;

void\* poolingDesc;

#endif // CUDNN

//#endif // GPU

};

// network.h

typedef enum {

CONSTANT, STEP, EXP, POLY, STEPS, SIG, RANDOM, SGDR

} learning\_rate\_policy;

// network.h

typedef struct network {

int n;

int batch;

uint64\_t \*seen;

int \*cur\_iteration;

float loss\_scale;

int \*t;

float epoch;

int subdivisions;

layer \*layers;

float \*output;

learning\_rate\_policy policy;

int benchmark\_layers;

float learning\_rate;

float learning\_rate\_min;

float learning\_rate\_max;

int batches\_per\_cycle;

int batches\_cycle\_mult;

float momentum;

float decay;

float gamma;

float scale;

float power;

int time\_steps;

int step;

int max\_batches;

int num\_boxes;

int train\_images\_num;

float \*seq\_scales;

float \*scales;

int \*steps;

int num\_steps;

int burn\_in;

int cudnn\_half;

int adam;

float B1;

float B2;

float eps;

int inputs;

int outputs;

int truths;

int notruth;

int h, w, c;

int max\_crop;

int min\_crop;

float max\_ratio;

float min\_ratio;

int center;

int flip; // horizontal flip 50% probability augmentaiont for classifier training (default = 1)

int gaussian\_noise;

int blur;

int mixup;

float label\_smooth\_eps;

int resize\_step;

int attention;

int adversarial;

float adversarial\_lr;

int letter\_box;

float angle;

float aspect;

float exposure;

float saturation;

float hue;

int random;

int track;

int augment\_speed;

int sequential\_subdivisions;

int init\_sequential\_subdivisions;

int current\_subdivision;

int try\_fix\_nan;

int gpu\_index;

tree \*hierarchy;

float \*input;

float \*truth;

float \*delta;

float \*workspace;

int train;

int index;

float \*cost;

float clip;

//#ifdef GPU

//float \*input\_gpu;

//float \*truth\_gpu;

float \*delta\_gpu;

float \*output\_gpu;

float \*input\_state\_gpu;

float \*input\_pinned\_cpu;

int input\_pinned\_cpu\_flag;

float \*\*input\_gpu;

float \*\*truth\_gpu;

float \*\*input16\_gpu;

float \*\*output16\_gpu;

size\_t \*max\_input16\_size;

size\_t \*max\_output16\_size;

int wait\_stream;

float \*global\_delta\_gpu;

float \*state\_delta\_gpu;

size\_t max\_delta\_gpu\_size;

//#endif // GPU

int optimized\_memory;

int dynamic\_minibatch;

size\_t workspace\_size\_limit;

} network;

// network.h

typedef struct network\_state {

float \*truth;

float \*input;

float \*delta;

float \*workspace;

int train;

int index;

network net;

} network\_state;

//typedef struct {

// int w;

// int h;

// float scale;

// float rad;

// float dx;

// float dy;

// float aspect;

//} augment\_args;

// image.h

typedef struct image {

int w;

int h;

int c;

float \*data;

} image;

//typedef struct {

// int w;

// int h;

// int c;

// float \*data;

//} image;

// box.h

typedef struct box {

float x, y, w, h;

} box;

// box.h

typedef struct boxabs {

float left, right, top, bot;

} boxabs;

// box.h

typedef struct dxrep {

float dt, db, dl, dr;

} dxrep;

// box.h

typedef struct ious {

float iou, giou, diou, ciou;

dxrep dx\_iou;

dxrep dx\_giou;

} ious;

// box.h

typedef struct detection{

box bbox;

int classes;

float \*prob;

float \*mask;

float objectness;

int sort\_class;

float \*uc; // Gaussian\_YOLOv3 - tx,ty,tw,th uncertainty

int points; // bit-0 - center, bit-1 - top-left-corner, bit-2 - bottom-right-corner

} detection;

// network.c -batch inference

typedef struct det\_num\_pair {

int num;

detection \*dets;

} det\_num\_pair, \*pdet\_num\_pair;

// matrix.h

typedef struct matrix {

int rows, cols;

float \*\*vals;

} matrix;

// data.h

typedef struct data {

int w, h;

matrix X;

matrix y;

int shallow;

int \*num\_boxes;

box \*\*boxes;

} data;

// data.h

typedef enum {

CLASSIFICATION\_DATA, DETECTION\_DATA, CAPTCHA\_DATA, REGION\_DATA, IMAGE\_DATA, COMPARE\_DATA, WRITING\_DATA, SWAG\_DATA, TAG\_DATA, OLD\_CLASSIFICATION\_DATA, STUDY\_DATA, DET\_DATA, SUPER\_DATA, LETTERBOX\_DATA, REGRESSION\_DATA, SEGMENTATION\_DATA, INSTANCE\_DATA, ISEG\_DATA

} data\_type;

// data.h

typedef struct load\_args {

int threads;

char \*\*paths;

char \*path;

int n;

int m;

char \*\*labels;

int h;

int w;

int c; // color depth

int out\_w;

int out\_h;

int nh;

int nw;

int num\_boxes;

int min, max, size;

int classes;

int background;

int scale;

int center;

int coords;

int mini\_batch;

int track;

int augment\_speed;

int letter\_box;

int show\_imgs;

int dontuse\_opencv;

float jitter;

int flip;

int gaussian\_noise;

int blur;

int mixup;

float label\_smooth\_eps;

float angle;

float aspect;

float saturation;

float exposure;

float hue;

data \*d;

image \*im;

image \*resized;

data\_type type;

tree \*hierarchy;

} load\_args;

// data.h

typedef struct box\_label {

int id;

float x, y, w, h;

float left, right, top, bottom;

} box\_label;

// list.h

//typedef struct node {

// void \*val;

// struct node \*next;

// struct node \*prev;

//} node;

// list.h

//typedef struct list {

// int size;

// node \*front;

// node \*back;

//} list;

// -----------------------------------------------------

// parser.c

LIB\_API network \*load\_network(char \*cfg, char \*weights, int clear);

LIB\_API network \*load\_network\_custom(char \*cfg, char \*weights, int clear, int batch);

LIB\_API network \*load\_network(char \*cfg, char \*weights, int clear);

LIB\_API void free\_network(network net);

// network.c

LIB\_API load\_args get\_base\_args(network \*net);

// box.h

LIB\_API void do\_nms\_sort(detection \*dets, int total, int classes, float thresh);

LIB\_API void do\_nms\_obj(detection \*dets, int total, int classes, float thresh);

LIB\_API void diounms\_sort(detection \*dets, int total, int classes, float thresh, NMS\_KIND nms\_kind, float beta1);

// network.h

LIB\_API float \*network\_predict(network net, float \*input);

LIB\_API float \*network\_predict\_ptr(network \*net, float \*input);

LIB\_API detection \*get\_network\_boxes(network \*net, int w, int h, float thresh, float hier, int \*map, int relative, int \*num, int letter);

LIB\_API det\_num\_pair\* network\_predict\_batch(network \*net, image im, int batch\_size, int w, int h, float thresh, float hier, int \*map, int relative, int letter);

LIB\_API void free\_detections(detection \*dets, int n);

LIB\_API void free\_batch\_detections(det\_num\_pair \*det\_num\_pairs, int n);

LIB\_API void fuse\_conv\_batchnorm(network net);

LIB\_API void calculate\_binary\_weights(network net);

LIB\_API char \*detection\_to\_json(detection \*dets, int nboxes, int classes, char \*\*names, long long int frame\_id, char \*filename);

LIB\_API layer\* get\_network\_layer(network\* net, int i);

//LIB\_API detection \*get\_network\_boxes(network \*net, int w, int h, float thresh, float hier, int \*map, int relative, int \*num, int letter);

LIB\_API detection \*make\_network\_boxes(network \*net, float thresh, int \*num);

LIB\_API void reset\_rnn(network \*net);

LIB\_API float \*network\_predict\_image(network \*net, image im);

LIB\_API float \*network\_predict\_image\_letterbox(network \*net, image im);

LIB\_API float validate\_detector\_map(char \*datacfg, char \*cfgfile, char \*weightfile, float thresh\_calc\_avg\_iou, const float iou\_thresh, const int map\_points, int letter\_box, network \*existing\_net);

LIB\_API void train\_detector(char \*datacfg, char \*cfgfile, char \*weightfile, int \*gpus, int ngpus, int clear, int dont\_show, int calc\_map, int mjpeg\_port, int show\_imgs, int benchmark\_layers, char\* chart\_path);

LIB\_API void test\_detector(char \*datacfg, char \*cfgfile, char \*weightfile, char \*filename, float thresh,

float hier\_thresh, int dont\_show, int ext\_output, int save\_labels, char \*outfile, int letter\_box, int benchmark\_layers);

LIB\_API int network\_width(network \*net);

LIB\_API int network\_height(network \*net);

LIB\_API void optimize\_picture(network \*net, image orig, int max\_layer, float scale, float rate, float thresh, int norm);

// image.h

LIB\_API void make\_image\_red(image im);

LIB\_API image make\_attention\_image(int img\_size, float \*original\_delta\_cpu, float \*original\_input\_cpu, int w, int h, int c);

LIB\_API image resize\_image(image im, int w, int h);

LIB\_API void quantize\_image(image im);

LIB\_API void copy\_image\_from\_bytes(image im, char \*pdata);

LIB\_API image letterbox\_image(image im, int w, int h);

LIB\_API void rgbgr\_image(image im);

LIB\_API image make\_image(int w, int h, int c);

LIB\_API image load\_image\_color(char \*filename, int w, int h);

LIB\_API void free\_image(image m);

LIB\_API image crop\_image(image im, int dx, int dy, int w, int h);

LIB\_API image resize\_min(image im, int min);

// layer.h

LIB\_API void free\_layer\_custom(layer l, int keep\_cudnn\_desc);

LIB\_API void free\_layer(layer l);

// data.c

LIB\_API void free\_data(data d);

LIB\_API pthread\_t load\_data(load\_args args);

LIB\_API void free\_load\_threads(void \*ptr);

LIB\_API pthread\_t load\_data\_in\_thread(load\_args args);

LIB\_API void \*load\_thread(void \*ptr);

// dark\_cuda.h

LIB\_API void cuda\_pull\_array(float \*x\_gpu, float \*x, size\_t n);

LIB\_API void cuda\_pull\_array\_async(float \*x\_gpu, float \*x, size\_t n);

LIB\_API void cuda\_set\_device(int n);

LIB\_API void \*cuda\_get\_context();

// utils.h

LIB\_API void free\_ptrs(void \*\*ptrs, int n);

LIB\_API void top\_k(float \*a, int n, int k, int \*index);

// tree.h

LIB\_API tree \*read\_tree(char \*filename);

// option\_list.h

LIB\_API metadata get\_metadata(char \*file);

// http\_stream.h

LIB\_API void delete\_json\_sender();

LIB\_API void send\_json\_custom(char const\* send\_buf, int port, int timeout);

LIB\_API double get\_time\_point();

void start\_timer();

void stop\_timer();

double get\_time();

void stop\_timer\_and\_show();

void stop\_timer\_and\_show\_name(char \*name);

void show\_total\_time();

// gemm.h

LIB\_API void init\_cpu();

#ifdef \_\_cplusplus

}

#endif // \_\_cplusplus

#endif // DARKNET\_API