// Oh boy, why am I about to do this....

#ifndef NETWORK\_H

#define NETWORK\_H

#include "darknet.h"

#include <stdint.h>

#include "layer.h"

#include "image.h"

#include "data.h"

#include "tree.h"

#ifdef \_\_cplusplus

extern "C" {

#endif

/\*

typedef enum {

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

} learning\_rate\_policy;

typedef struct network{

float \*workspace;

int n;

int batch;

uint64\_t \*seen;

float epoch;

int subdivisions;

float momentum;

float decay;

layer \*layers;

int outputs;

float \*output;

learning\_rate\_policy policy;

float learning\_rate;

float gamma;

float scale;

float power;

int time\_steps;

int step;

int max\_batches;

float \*scales;

int \*steps;

int num\_steps;

int burn\_in;

int cudnn\_half;

int adam;

float B1;

float B2;

float eps;

int inputs;

int h, w, c;

int max\_crop;

int min\_crop;

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

float angle;

float aspect;

float exposure;

float saturation;

float hue;

int small\_object;

int gpu\_index;

tree \*hierarchy;

#ifdef GPU

float \*input\_state\_gpu;

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;

#endif

} network;

typedef struct network\_state {

float \*truth;

float \*input;

float \*delta;

float \*workspace;

int train;

int index;

network net;

} network\_state;

\*/

#ifdef GPU

float train\_networks(network \*nets, int n, data d, int interval);

void sync\_nets(network \*nets, int n, int interval);

float train\_network\_datum\_gpu(network net, float \*x, float \*y);

float \*network\_predict\_gpu(network net, float \*input);

float \* get\_network\_output\_gpu\_layer(network net, int i);

float \* get\_network\_delta\_gpu\_layer(network net, int i);

float \*get\_network\_output\_gpu(network net);

void forward\_network\_gpu(network net, network\_state state);

void backward\_network\_gpu(network net, network\_state state);

void update\_network\_gpu(network net);

void forward\_backward\_network\_gpu(network net, float \*x, float \*y);

#endif

float get\_current\_seq\_subdivisions(network net);

int get\_sequence\_value(network net);

float get\_current\_rate(network net);

int get\_current\_batch(network net);

int64\_t get\_current\_iteration(network net);

//void free\_network(network net); // darknet.h

void compare\_networks(network n1, network n2, data d);

char \*get\_layer\_string(LAYER\_TYPE a);

network make\_network(int n);

void forward\_network(network net, network\_state state);

void backward\_network(network net, network\_state state);

void update\_network(network net);

float train\_network(network net, data d);

float train\_network\_waitkey(network net, data d, int wait\_key);

float train\_network\_batch(network net, data d, int n);

float train\_network\_sgd(network net, data d, int n);

float train\_network\_datum(network net, float \*x, float \*y);

matrix network\_predict\_data(network net, data test);

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

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

float network\_accuracy(network net, data d);

float \*network\_accuracies(network net, data d, int n);

float network\_accuracy\_multi(network net, data d, int n);

void top\_predictions(network net, int n, int \*index);

float \*get\_network\_output(network net);

float \*get\_network\_output\_layer(network net, int i);

float \*get\_network\_delta\_layer(network net, int i);

float \*get\_network\_delta(network net);

int get\_network\_output\_size\_layer(network net, int i);

int get\_network\_output\_size(network net);

image get\_network\_image(network net);

image get\_network\_image\_layer(network net, int i);

int get\_predicted\_class\_network(network net);

void print\_network(network net);

void visualize\_network(network net);

int resize\_network(network \*net, int w, int h);

void set\_batch\_network(network \*net, int b);

int get\_network\_input\_size(network net);

float get\_network\_cost(network net);

//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 free\_detections(detection \*dets, int n);

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

//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 float \*network\_predict\_image(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, 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);

//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);

int get\_network\_nuisance(network net);

int get\_network\_background(network net);

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

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

network combine\_train\_valid\_networks(network net\_train, network net\_map);

void copy\_weights\_net(network net\_train, network \*net\_map);

void free\_network\_recurrent\_state(network net);

void randomize\_network\_recurrent\_state(network net);

void remember\_network\_recurrent\_state(network net);

void restore\_network\_recurrent\_state(network net);

#ifdef \_\_cplusplus

}

#endif

#endif