#include "darknet.h"

#include <time.h>

#include <stdlib.h>

#include <stdio.h>

#if defined(\_MSC\_VER) && defined(\_DEBUG)

#include <crtdbg.h>

#endif

#include "parser.h"

#include "utils.h"

#include "dark\_cuda.h"

#include "blas.h"

#include "connected\_layer.h"

extern void predict\_classifier(char \*datacfg, char \*cfgfile, char \*weightfile, char \*filename, int top);

extern void run\_voxel(int argc, char \*\*argv);

extern void run\_yolo(int argc, char \*\*argv);

extern void run\_detector(int argc, char \*\*argv);

extern void run\_coco(int argc, char \*\*argv);

extern void run\_writing(int argc, char \*\*argv);

extern void run\_captcha(int argc, char \*\*argv);

extern void run\_nightmare(int argc, char \*\*argv);

extern void run\_dice(int argc, char \*\*argv);

extern void run\_compare(int argc, char \*\*argv);

extern void run\_classifier(int argc, char \*\*argv);

extern void run\_char\_rnn(int argc, char \*\*argv);

extern void run\_vid\_rnn(int argc, char \*\*argv);

extern void run\_tag(int argc, char \*\*argv);

extern void run\_cifar(int argc, char \*\*argv);

extern void run\_go(int argc, char \*\*argv);

extern void run\_art(int argc, char \*\*argv);

extern void run\_super(int argc, char \*\*argv);

void average(int argc, char \*argv[])

{

char \*cfgfile = argv[2];

char \*outfile = argv[3];

gpu\_index = -1;

network net = parse\_network\_cfg(cfgfile);

network sum = parse\_network\_cfg(cfgfile);

char \*weightfile = argv[4];

load\_weights(&sum, weightfile);

int i, j;

int n = argc - 5;

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

weightfile = argv[i+5];

load\_weights(&net, weightfile);

for(j = 0; j < net.n; ++j){

layer l = net.layers[j];

layer out = sum.layers[j];

if(l.type == CONVOLUTIONAL){

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

axpy\_cpu(l.n, 1, l.biases, 1, out.biases, 1);

axpy\_cpu(num, 1, l.weights, 1, out.weights, 1);

if(l.batch\_normalize){

axpy\_cpu(l.n, 1, l.scales, 1, out.scales, 1);

axpy\_cpu(l.n, 1, l.rolling\_mean, 1, out.rolling\_mean, 1);

axpy\_cpu(l.n, 1, l.rolling\_variance, 1, out.rolling\_variance, 1);

}

}

if(l.type == CONNECTED){

axpy\_cpu(l.outputs, 1, l.biases, 1, out.biases, 1);

axpy\_cpu(l.outputs\*l.inputs, 1, l.weights, 1, out.weights, 1);

}

}

}

n = n+1;

for(j = 0; j < net.n; ++j){

layer l = sum.layers[j];

if(l.type == CONVOLUTIONAL){

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

scal\_cpu(l.n, 1./n, l.biases, 1);

scal\_cpu(num, 1./n, l.weights, 1);

if(l.batch\_normalize){

scal\_cpu(l.n, 1./n, l.scales, 1);

scal\_cpu(l.n, 1./n, l.rolling\_mean, 1);

scal\_cpu(l.n, 1./n, l.rolling\_variance, 1);

}

}

if(l.type == CONNECTED){

scal\_cpu(l.outputs, 1./n, l.biases, 1);

scal\_cpu(l.outputs\*l.inputs, 1./n, l.weights, 1);

}

}

save\_weights(sum, outfile);

}

void speed(char \*cfgfile, int tics)

{

if (tics == 0) tics = 1000;

network net = parse\_network\_cfg(cfgfile);

set\_batch\_network(&net, 1);

int i;

time\_t start = time(0);

image im = make\_image(net.w, net.h, net.c);

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

network\_predict(net, im.data);

}

double t = difftime(time(0), start);

printf("\n%d evals, %f Seconds\n", tics, t);

printf("Speed: %f sec/eval\n", t/tics);

printf("Speed: %f Hz\n", tics/t);

}

void operations(char \*cfgfile)

{

gpu\_index = -1;

network net = parse\_network\_cfg(cfgfile);

int i;

long ops = 0;

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

layer l = net.layers[i];

if(l.type == CONVOLUTIONAL){

ops += 2l \* l.n \* l.size\*l.size\*l.c \* l.out\_h\*l.out\_w;

} else if(l.type == CONNECTED){

ops += 2l \* l.inputs \* l.outputs;

} else if (l.type == RNN){

ops += 2l \* l.input\_layer->inputs \* l.input\_layer->outputs;

ops += 2l \* l.self\_layer->inputs \* l.self\_layer->outputs;

ops += 2l \* l.output\_layer->inputs \* l.output\_layer->outputs;

} else if (l.type == GRU){

ops += 2l \* l.uz->inputs \* l.uz->outputs;

ops += 2l \* l.uh->inputs \* l.uh->outputs;

ops += 2l \* l.ur->inputs \* l.ur->outputs;

ops += 2l \* l.wz->inputs \* l.wz->outputs;

ops += 2l \* l.wh->inputs \* l.wh->outputs;

ops += 2l \* l.wr->inputs \* l.wr->outputs;

} else if (l.type == LSTM){

ops += 2l \* l.uf->inputs \* l.uf->outputs;

ops += 2l \* l.ui->inputs \* l.ui->outputs;

ops += 2l \* l.ug->inputs \* l.ug->outputs;

ops += 2l \* l.uo->inputs \* l.uo->outputs;

ops += 2l \* l.wf->inputs \* l.wf->outputs;

ops += 2l \* l.wi->inputs \* l.wi->outputs;

ops += 2l \* l.wg->inputs \* l.wg->outputs;

ops += 2l \* l.wo->inputs \* l.wo->outputs;

}

}

printf("Floating Point Operations: %ld\n", ops);

printf("Floating Point Operations: %.2f Bn\n", (float)ops/1000000000.);

}

void oneoff(char \*cfgfile, char \*weightfile, char \*outfile)

{

gpu\_index = -1;

network net = parse\_network\_cfg(cfgfile);

int oldn = net.layers[net.n - 2].n;

int c = net.layers[net.n - 2].c;

net.layers[net.n - 2].n = 9372;

net.layers[net.n - 2].biases += 5;

net.layers[net.n - 2].weights += 5\*c;

if(weightfile){

load\_weights(&net, weightfile);

}

net.layers[net.n - 2].biases -= 5;

net.layers[net.n - 2].weights -= 5\*c;

net.layers[net.n - 2].n = oldn;

printf("%d\n", oldn);

layer l = net.layers[net.n - 2];

copy\_cpu(l.n/3, l.biases, 1, l.biases + l.n/3, 1);

copy\_cpu(l.n/3, l.biases, 1, l.biases + 2\*l.n/3, 1);

copy\_cpu(l.n/3\*l.c, l.weights, 1, l.weights + l.n/3\*l.c, 1);

copy\_cpu(l.n/3\*l.c, l.weights, 1, l.weights + 2\*l.n/3\*l.c, 1);

\*net.seen = 0;

\*net.cur\_iteration = 0;

save\_weights(net, outfile);

}

void partial(char \*cfgfile, char \*weightfile, char \*outfile, int max)

{

gpu\_index = -1;

network net = parse\_network\_cfg\_custom(cfgfile, 1, 1);

if(weightfile){

load\_weights\_upto(&net, weightfile, max);

}

\*net.seen = 0;

\*net.cur\_iteration = 0;

save\_weights\_upto(net, outfile, max);

}

#include "convolutional\_layer.h"

void rescale\_net(char \*cfgfile, char \*weightfile, char \*outfile)

{

gpu\_index = -1;

network net = parse\_network\_cfg(cfgfile);

if(weightfile){

load\_weights(&net, weightfile);

}

int i;

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

layer l = net.layers[i];

if(l.type == CONVOLUTIONAL){

rescale\_weights(l, 2, -.5);

break;

}

}

save\_weights(net, outfile);

}

void rgbgr\_net(char \*cfgfile, char \*weightfile, char \*outfile)

{

gpu\_index = -1;

network net = parse\_network\_cfg(cfgfile);

if(weightfile){

load\_weights(&net, weightfile);

}

int i;

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

layer l = net.layers[i];

if(l.type == CONVOLUTIONAL){

rgbgr\_weights(l);

break;

}

}

save\_weights(net, outfile);

}

void reset\_normalize\_net(char \*cfgfile, char \*weightfile, char \*outfile)

{

gpu\_index = -1;

network net = parse\_network\_cfg(cfgfile);

if (weightfile) {

load\_weights(&net, weightfile);

}

int i;

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

layer l = net.layers[i];

if (l.type == CONVOLUTIONAL && l.batch\_normalize) {

denormalize\_convolutional\_layer(l);

}

if (l.type == CONNECTED && l.batch\_normalize) {

denormalize\_connected\_layer(l);

}

if (l.type == GRU && l.batch\_normalize) {

denormalize\_connected\_layer(\*l.input\_z\_layer);

denormalize\_connected\_layer(\*l.input\_r\_layer);

denormalize\_connected\_layer(\*l.input\_h\_layer);

denormalize\_connected\_layer(\*l.state\_z\_layer);

denormalize\_connected\_layer(\*l.state\_r\_layer);

denormalize\_connected\_layer(\*l.state\_h\_layer);

}

if (l.type == LSTM && l.batch\_normalize) {

denormalize\_connected\_layer(\*l.wf);

denormalize\_connected\_layer(\*l.wi);

denormalize\_connected\_layer(\*l.wg);

denormalize\_connected\_layer(\*l.wo);

denormalize\_connected\_layer(\*l.uf);

denormalize\_connected\_layer(\*l.ui);

denormalize\_connected\_layer(\*l.ug);

denormalize\_connected\_layer(\*l.uo);

}

}

save\_weights(net, outfile);

}

layer normalize\_layer(layer l, int n)

{

int j;

l.batch\_normalize=1;

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

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

l.scales[j] = 1;

}

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

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

return l;

}

void normalize\_net(char \*cfgfile, char \*weightfile, char \*outfile)

{

gpu\_index = -1;

network net = parse\_network\_cfg(cfgfile);

if(weightfile){

load\_weights(&net, weightfile);

}

int i;

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

layer l = net.layers[i];

if(l.type == CONVOLUTIONAL && !l.batch\_normalize){

net.layers[i] = normalize\_layer(l, l.n);

}

if (l.type == CONNECTED && !l.batch\_normalize) {

net.layers[i] = normalize\_layer(l, l.outputs);

}

if (l.type == GRU && l.batch\_normalize) {

\*l.input\_z\_layer = normalize\_layer(\*l.input\_z\_layer, l.input\_z\_layer->outputs);

\*l.input\_r\_layer = normalize\_layer(\*l.input\_r\_layer, l.input\_r\_layer->outputs);

\*l.input\_h\_layer = normalize\_layer(\*l.input\_h\_layer, l.input\_h\_layer->outputs);

\*l.state\_z\_layer = normalize\_layer(\*l.state\_z\_layer, l.state\_z\_layer->outputs);

\*l.state\_r\_layer = normalize\_layer(\*l.state\_r\_layer, l.state\_r\_layer->outputs);

\*l.state\_h\_layer = normalize\_layer(\*l.state\_h\_layer, l.state\_h\_layer->outputs);

net.layers[i].batch\_normalize=1;

}

if (l.type == LSTM && l.batch\_normalize) {

\*l.wf = normalize\_layer(\*l.wf, l.wf->outputs);

\*l.wi = normalize\_layer(\*l.wi, l.wi->outputs);

\*l.wg = normalize\_layer(\*l.wg, l.wg->outputs);

\*l.wo = normalize\_layer(\*l.wo, l.wo->outputs);

\*l.uf = normalize\_layer(\*l.uf, l.uf->outputs);

\*l.ui = normalize\_layer(\*l.ui, l.ui->outputs);

\*l.ug = normalize\_layer(\*l.ug, l.ug->outputs);

\*l.uo = normalize\_layer(\*l.uo, l.uo->outputs);

net.layers[i].batch\_normalize=1;

}

}

save\_weights(net, outfile);

}

void statistics\_net(char \*cfgfile, char \*weightfile)

{

gpu\_index = -1;

network net = parse\_network\_cfg(cfgfile);

if (weightfile) {

load\_weights(&net, weightfile);

}

int i;

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

layer l = net.layers[i];

if (l.type == CONNECTED && l.batch\_normalize) {

printf("Connected Layer %d\n", i);

statistics\_connected\_layer(l);

}

if (l.type == GRU && l.batch\_normalize) {

printf("GRU Layer %d\n", i);

printf("Input Z\n");

statistics\_connected\_layer(\*l.input\_z\_layer);

printf("Input R\n");

statistics\_connected\_layer(\*l.input\_r\_layer);

printf("Input H\n");

statistics\_connected\_layer(\*l.input\_h\_layer);

printf("State Z\n");

statistics\_connected\_layer(\*l.state\_z\_layer);

printf("State R\n");

statistics\_connected\_layer(\*l.state\_r\_layer);

printf("State H\n");

statistics\_connected\_layer(\*l.state\_h\_layer);

}

if (l.type == LSTM && l.batch\_normalize) {

printf("LSTM Layer %d\n", i);

printf("wf\n");

statistics\_connected\_layer(\*l.wf);

printf("wi\n");

statistics\_connected\_layer(\*l.wi);

printf("wg\n");

statistics\_connected\_layer(\*l.wg);

printf("wo\n");

statistics\_connected\_layer(\*l.wo);

printf("uf\n");

statistics\_connected\_layer(\*l.uf);

printf("ui\n");

statistics\_connected\_layer(\*l.ui);

printf("ug\n");

statistics\_connected\_layer(\*l.ug);

printf("uo\n");

statistics\_connected\_layer(\*l.uo);

}

printf("\n");

}

}

void denormalize\_net(char \*cfgfile, char \*weightfile, char \*outfile)

{

gpu\_index = -1;

network net = parse\_network\_cfg(cfgfile);

if (weightfile) {

load\_weights(&net, weightfile);

}

int i;

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

layer l = net.layers[i];

if (l.type == CONVOLUTIONAL && l.batch\_normalize) {

denormalize\_convolutional\_layer(l);

net.layers[i].batch\_normalize=0;

}

if (l.type == CONNECTED && l.batch\_normalize) {

denormalize\_connected\_layer(l);

net.layers[i].batch\_normalize=0;

}

if (l.type == GRU && l.batch\_normalize) {

denormalize\_connected\_layer(\*l.input\_z\_layer);

denormalize\_connected\_layer(\*l.input\_r\_layer);

denormalize\_connected\_layer(\*l.input\_h\_layer);

denormalize\_connected\_layer(\*l.state\_z\_layer);

denormalize\_connected\_layer(\*l.state\_r\_layer);

denormalize\_connected\_layer(\*l.state\_h\_layer);

l.input\_z\_layer->batch\_normalize = 0;

l.input\_r\_layer->batch\_normalize = 0;

l.input\_h\_layer->batch\_normalize = 0;

l.state\_z\_layer->batch\_normalize = 0;

l.state\_r\_layer->batch\_normalize = 0;

l.state\_h\_layer->batch\_normalize = 0;

net.layers[i].batch\_normalize=0;

}

if (l.type == GRU && l.batch\_normalize) {

denormalize\_connected\_layer(\*l.wf);

denormalize\_connected\_layer(\*l.wi);

denormalize\_connected\_layer(\*l.wg);

denormalize\_connected\_layer(\*l.wo);

denormalize\_connected\_layer(\*l.uf);

denormalize\_connected\_layer(\*l.ui);

denormalize\_connected\_layer(\*l.ug);

denormalize\_connected\_layer(\*l.uo);

l.wf->batch\_normalize = 0;

l.wi->batch\_normalize = 0;

l.wg->batch\_normalize = 0;

l.wo->batch\_normalize = 0;

l.uf->batch\_normalize = 0;

l.ui->batch\_normalize = 0;

l.ug->batch\_normalize = 0;

l.uo->batch\_normalize = 0;

net.layers[i].batch\_normalize=0;

}

}

save\_weights(net, outfile);

}

void visualize(char \*cfgfile, char \*weightfile)

{

network net = parse\_network\_cfg(cfgfile);

if(weightfile){

load\_weights(&net, weightfile);

}

visualize\_network(net);

#ifdef OPENCV

wait\_until\_press\_key\_cv();

#endif

}

int main(int argc, char \*\*argv)

{

#ifdef \_DEBUG

\_CrtSetDbgFlag(\_CRTDBG\_ALLOC\_MEM\_DF | \_CRTDBG\_LEAK\_CHECK\_DF);

#endif

int i;

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

if (!argv[i]) continue;

strip\_args(argv[i]);

}

//test\_resize("data/bad.jpg");

//test\_box();

//test\_convolutional\_layer();

if(argc < 2){

fprintf(stderr, "usage: %s <function>\n", argv[0]);

return 0;

}

gpu\_index = find\_int\_arg(argc, argv, "-i", 0);

if(find\_arg(argc, argv, "-nogpu")) {

gpu\_index = -1;

printf("\n Currently Darknet doesn't support -nogpu flag. If you want to use CPU - please compile Darknet with GPU=0 in the Makefile, or compile darknet\_no\_gpu.sln on Windows.\n");

exit(-1);

}

#ifndef GPU

gpu\_index = -1;

printf(" GPU isn't used \n");

init\_cpu();

#else // GPU

if(gpu\_index >= 0){

cuda\_set\_device(gpu\_index);

CHECK\_CUDA(cudaSetDeviceFlags(cudaDeviceScheduleBlockingSync));

}

show\_cuda\_cudnn\_info();

cuda\_debug\_sync = find\_arg(argc, argv, "-cuda\_debug\_sync");

#ifdef CUDNN\_HALF

printf(" CUDNN\_HALF=1 \n");

#endif // CUDNN\_HALF

#endif // GPU

show\_opencv\_info();

if (0 == strcmp(argv[1], "average")){

average(argc, argv);

} else if (0 == strcmp(argv[1], "yolo")){

run\_yolo(argc, argv);

} else if (0 == strcmp(argv[1], "voxel")){

run\_voxel(argc, argv);

} else if (0 == strcmp(argv[1], "super")){

run\_super(argc, argv);

} else if (0 == strcmp(argv[1], "detector")){

run\_detector(argc, argv);

} else if (0 == strcmp(argv[1], "detect")){

float thresh = find\_float\_arg(argc, argv, "-thresh", .24);

int ext\_output = find\_arg(argc, argv, "-ext\_output");

char \*filename = (argc > 4) ? argv[4]: 0;

test\_detector("cfg/coco.data", argv[2], argv[3], filename, thresh, 0.5, 0, ext\_output, 0, NULL, 0, 0);

} else if (0 == strcmp(argv[1], "cifar")){

run\_cifar(argc, argv);

} else if (0 == strcmp(argv[1], "go")){

run\_go(argc, argv);

} else if (0 == strcmp(argv[1], "rnn")){

run\_char\_rnn(argc, argv);

} else if (0 == strcmp(argv[1], "vid")){

run\_vid\_rnn(argc, argv);

} else if (0 == strcmp(argv[1], "coco")){

run\_coco(argc, argv);

} else if (0 == strcmp(argv[1], "classify")){

predict\_classifier("cfg/imagenet1k.data", argv[2], argv[3], argv[4], 5);

} else if (0 == strcmp(argv[1], "classifier")){

run\_classifier(argc, argv);

} else if (0 == strcmp(argv[1], "art")){

run\_art(argc, argv);

} else if (0 == strcmp(argv[1], "tag")){

run\_tag(argc, argv);

} else if (0 == strcmp(argv[1], "compare")){

run\_compare(argc, argv);

} else if (0 == strcmp(argv[1], "dice")){

run\_dice(argc, argv);

} else if (0 == strcmp(argv[1], "writing")){

run\_writing(argc, argv);

} else if (0 == strcmp(argv[1], "3d")){

composite\_3d(argv[2], argv[3], argv[4], (argc > 5) ? atof(argv[5]) : 0);

} else if (0 == strcmp(argv[1], "test")){

test\_resize(argv[2]);

} else if (0 == strcmp(argv[1], "captcha")){

run\_captcha(argc, argv);

} else if (0 == strcmp(argv[1], "nightmare")){

run\_nightmare(argc, argv);

} else if (0 == strcmp(argv[1], "rgbgr")){

rgbgr\_net(argv[2], argv[3], argv[4]);

} else if (0 == strcmp(argv[1], "reset")){

reset\_normalize\_net(argv[2], argv[3], argv[4]);

} else if (0 == strcmp(argv[1], "denormalize")){

denormalize\_net(argv[2], argv[3], argv[4]);

} else if (0 == strcmp(argv[1], "statistics")){

statistics\_net(argv[2], argv[3]);

} else if (0 == strcmp(argv[1], "normalize")){

normalize\_net(argv[2], argv[3], argv[4]);

} else if (0 == strcmp(argv[1], "rescale")){

rescale\_net(argv[2], argv[3], argv[4]);

} else if (0 == strcmp(argv[1], "ops")){

operations(argv[2]);

} else if (0 == strcmp(argv[1], "speed")){

speed(argv[2], (argc > 3 && argv[3]) ? atoi(argv[3]) : 0);

} else if (0 == strcmp(argv[1], "oneoff")){

oneoff(argv[2], argv[3], argv[4]);

} else if (0 == strcmp(argv[1], "partial")){

partial(argv[2], argv[3], argv[4], atoi(argv[5]));

} else if (0 == strcmp(argv[1], "visualize")){

visualize(argv[2], (argc > 3) ? argv[3] : 0);

} else if (0 == strcmp(argv[1], "imtest")){

test\_resize(argv[2]);

} else {

fprintf(stderr, "Not an option: %s\n", argv[1]);

}

return 0;

}