#include "network.h"

#include "utils.h"

#include "parser.h"

#include "option\_list.h"

#include "blas.h"

#include "assert.h"

#include "classifier.h"

#include "dark\_cuda.h"

#ifdef WIN32

#include <time.h>

#include "gettimeofday.h"

#else

#include <sys/time.h>

#endif

float validate\_classifier\_single(char \*datacfg, char \*filename, char \*weightfile, network \*existing\_net, int topk\_custom);

float \*get\_regression\_values(char \*\*labels, int n)

{

float\* v = (float\*)xcalloc(n, sizeof(float));

int i;

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

char \*p = strchr(labels[i], ' ');

\*p = 0;

v[i] = atof(p+1);

}

return v;

}

void train\_classifier(char \*datacfg, char \*cfgfile, char \*weightfile, int \*gpus, int ngpus, int clear, int dontuse\_opencv, int dont\_show, int mjpeg\_port, int calc\_topk, int show\_imgs, char\* chart\_path)

{

int i;

float avg\_loss = -1;

char \*base = basecfg(cfgfile);

printf("%s\n", base);

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

network\* nets = (network\*)xcalloc(ngpus, sizeof(network));

srand(time(0));

int seed = rand();

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

srand(seed);

#ifdef GPU

cuda\_set\_device(gpus[i]);

#endif

nets[i] = parse\_network\_cfg(cfgfile);

if(weightfile){

load\_weights(&nets[i], weightfile);

}

if (clear) {

\*nets[i].seen = 0;

\*nets[i].cur\_iteration = 0;

}

nets[i].learning\_rate \*= ngpus;

}

srand(time(0));

network net = nets[0];

int imgs = net.batch \* net.subdivisions \* ngpus;

printf("Learning Rate: %g, Momentum: %g, Decay: %g\n", net.learning\_rate, net.momentum, net.decay);

list \*options = read\_data\_cfg(datacfg);

char \*backup\_directory = option\_find\_str(options, "backup", "/backup/");

char \*label\_list = option\_find\_str(options, "labels", "data/labels.list");

char \*train\_list = option\_find\_str(options, "train", "data/train.list");

int classes = option\_find\_int(options, "classes", 2);

int topk\_data = option\_find\_int(options, "top", 5);

char topk\_buff[10];

sprintf(topk\_buff, "top%d", topk\_data);

if (classes != net.layers[net.n - 1].inputs) {

printf("\n Error: num of filters = %d in the last conv-layer in cfg-file doesn't match to classes = %d in data-file \n",

net.layers[net.n - 1].inputs, classes);

getchar();

}

char \*\*labels = get\_labels(label\_list);

list \*plist = get\_paths(train\_list);

char \*\*paths = (char \*\*)list\_to\_array(plist);

printf("%d\n", plist->size);

int train\_images\_num = plist->size;

clock\_t time;

load\_args args = {0};

args.w = net.w;

args.h = net.h;

args.c = net.c;

args.threads = 32;

args.hierarchy = net.hierarchy;

args.dontuse\_opencv = dontuse\_opencv;

args.min = net.min\_crop;

args.max = net.max\_crop;

args.flip = net.flip;

args.blur = net.blur;

args.angle = net.angle;

args.aspect = net.aspect;

args.exposure = net.exposure;

args.saturation = net.saturation;

args.hue = net.hue;

args.size = net.w > net.h ? net.w : net.h;

args.label\_smooth\_eps = net.label\_smooth\_eps;

args.mixup = net.mixup;

if (dont\_show && show\_imgs) show\_imgs = 2;

args.show\_imgs = show\_imgs;

args.paths = paths;

args.classes = classes;

args.n = imgs;

args.m = train\_images\_num;

args.labels = labels;

args.type = CLASSIFICATION\_DATA;

#ifdef OPENCV

//args.threads = 3;

mat\_cv\* img = NULL;

float max\_img\_loss = 10;

int number\_of\_lines = 100;

int img\_size = 1000;

char windows\_name[100];

sprintf(windows\_name, "chart\_%s.png", base);

if (!dontuse\_opencv) img = draw\_train\_chart(windows\_name, max\_img\_loss, net.max\_batches, number\_of\_lines, img\_size, dont\_show, chart\_path);

#endif //OPENCV

data train;

data buffer;

pthread\_t load\_thread;

args.d = &buffer;

load\_thread = load\_data(args);

int iter\_save = get\_current\_batch(net);

int iter\_save\_last = get\_current\_batch(net);

int iter\_topk = get\_current\_batch(net);

float topk = 0;

int count = 0;

double start, time\_remaining, avg\_time = -1, alpha\_time = 0.01;

start = what\_time\_is\_it\_now();

while(get\_current\_batch(net) < net.max\_batches || net.max\_batches == 0){

time=clock();

pthread\_join(load\_thread, 0);

train = buffer;

load\_thread = load\_data(args);

printf("Loaded: %lf seconds\n", sec(clock()-time));

time=clock();

float loss = 0;

#ifdef GPU

if(ngpus == 1){

loss = train\_network(net, train);

} else {

loss = train\_networks(nets, ngpus, train, 4);

}

#else

loss = train\_network(net, train);

#endif

if(avg\_loss == -1 || isnan(avg\_loss) || isinf(avg\_loss)) avg\_loss = loss;

avg\_loss = avg\_loss\*.9 + loss\*.1;

i = get\_current\_batch(net);

int calc\_topk\_for\_each = iter\_topk + 2 \* train\_images\_num / (net.batch \* net.subdivisions); // calculate TOPk for each 2 Epochs

calc\_topk\_for\_each = fmax(calc\_topk\_for\_each, net.burn\_in);

calc\_topk\_for\_each = fmax(calc\_topk\_for\_each, 100);

if (i % 10 == 0) {

if (calc\_topk) {

fprintf(stderr, "\n (next TOP%d calculation at %d iterations) ", topk\_data, calc\_topk\_for\_each);

if (topk > 0) fprintf(stderr, " Last accuracy TOP%d = %2.2f %% \n", topk\_data, topk \* 100);

}

if (net.cudnn\_half) {

if (i < net.burn\_in \* 3) fprintf(stderr, " Tensor Cores are disabled until the first %d iterations are reached.\n", 3 \* net.burn\_in);

else fprintf(stderr, " Tensor Cores are used.\n");

}

}

int draw\_precision = 0;

if (calc\_topk && (i >= calc\_topk\_for\_each || i == net.max\_batches)) {

iter\_topk = i;

topk = validate\_classifier\_single(datacfg, cfgfile, weightfile, &net, topk\_data); // calc TOP-n

printf("\n accuracy %s = %f \n", topk\_buff, topk);

draw\_precision = 1;

}

time\_remaining = (net.max\_batches - i)\*(what\_time\_is\_it\_now() - start) / 60 / 60;

// set initial value, even if resume training from 10000 iteration

if (avg\_time < 0) avg\_time = time\_remaining;

else avg\_time = alpha\_time \* time\_remaining + (1 - alpha\_time) \* avg\_time;

start = what\_time\_is\_it\_now();

printf("%d, %.3f: %f, %f avg, %f rate, %lf seconds, %ld images, %f hours left\n", get\_current\_batch(net), (float)(\*net.seen)/ train\_images\_num, loss, avg\_loss, get\_current\_rate(net), sec(clock()-time), \*net.seen, avg\_time);

#ifdef OPENCV

if (!dontuse\_opencv) draw\_train\_loss(windows\_name, img, img\_size, avg\_loss, max\_img\_loss, i, net.max\_batches, topk, draw\_precision, topk\_buff, dont\_show, mjpeg\_port, avg\_time);

#endif // OPENCV

if (i >= (iter\_save + 1000)) {

iter\_save = i;

#ifdef GPU

if (ngpus != 1) sync\_nets(nets, ngpus, 0);

#endif

char buff[256];

sprintf(buff, "%s/%s\_%d.weights", backup\_directory, base, i);

save\_weights(net, buff);

}

if (i >= (iter\_save\_last + 100)) {

iter\_save\_last = i;

#ifdef GPU

if (ngpus != 1) sync\_nets(nets, ngpus, 0);

#endif

char buff[256];

sprintf(buff, "%s/%s\_last.weights", backup\_directory, base);

save\_weights(net, buff);

}

free\_data(train);

}

#ifdef GPU

if (ngpus != 1) sync\_nets(nets, ngpus, 0);

#endif

char buff[256];

sprintf(buff, "%s/%s\_final.weights", backup\_directory, base);

save\_weights(net, buff);

#ifdef OPENCV

release\_mat(&img);

destroy\_all\_windows\_cv();

#endif

pthread\_join(load\_thread, 0);

free\_data(buffer);

//free\_network(net);

for (i = 0; i < ngpus; ++i) free\_network(nets[i]);

free(nets);

//free\_ptrs((void\*\*)labels, classes);

free(labels);

free\_ptrs((void\*\*)paths, plist->size);

free\_list(plist);

free(nets);

free(base);

free\_list\_contents\_kvp(options);

free\_list(options);

}

/\*

void train\_classifier(char \*datacfg, char \*cfgfile, char \*weightfile, int clear)

{

srand(time(0));

float avg\_loss = -1;

char \*base = basecfg(cfgfile);

printf("%s\n", base);

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

if(weightfile){

load\_weights(&net, weightfile);

}

if(clear) \*net.seen = 0;

int imgs = net.batch \* net.subdivisions;

printf("Learning Rate: %g, Momentum: %g, Decay: %g\n", net.learning\_rate, net.momentum, net.decay);

list \*options = read\_data\_cfg(datacfg);

char \*backup\_directory = option\_find\_str(options, "backup", "/backup/");

char \*label\_list = option\_find\_str(options, "labels", "data/labels.list");

char \*train\_list = option\_find\_str(options, "train", "data/train.list");

int classes = option\_find\_int(options, "classes", 2);

char \*\*labels = get\_labels(label\_list);

list \*plist = get\_paths(train\_list);

char \*\*paths = (char \*\*)list\_to\_array(plist);

printf("%d\n", plist->size);

int N = plist->size;

clock\_t time;

load\_args args = {0};

args.w = net.w;

args.h = net.h;

args.threads = 8;

args.min = net.min\_crop;

args.max = net.max\_crop;

args.flip = net.flip;

args.angle = net.angle;

args.aspect = net.aspect;

args.exposure = net.exposure;

args.saturation = net.saturation;

args.hue = net.hue;

args.size = net.w;

args.hierarchy = net.hierarchy;

args.paths = paths;

args.classes = classes;

args.n = imgs;

args.m = N;

args.labels = labels;

args.type = CLASSIFICATION\_DATA;

data train;

data buffer;

pthread\_t load\_thread;

args.d = &buffer;

load\_thread = load\_data(args);

int epoch = (\*net.seen)/N;

while(get\_current\_batch(net) < net.max\_batches || net.max\_batches == 0){

time=clock();

pthread\_join(load\_thread, 0);

train = buffer;

load\_thread = load\_data(args);

printf("Loaded: %lf seconds\n", sec(clock()-time));

time=clock();

#ifdef OPENCV

if(0){

int u;

for(u = 0; u < imgs; ++u){

image im = float\_to\_image(net.w, net.h, 3, train.X.vals[u]);

show\_image(im, "loaded");

cvWaitKey(0);

}

}

#endif

float loss = train\_network(net, train);

free\_data(train);

if(avg\_loss == -1) avg\_loss = loss;

avg\_loss = avg\_loss\*.9 + loss\*.1;

printf("%d, %.3f: %f, %f avg, %f rate, %lf seconds, %d images\n", get\_current\_batch(net), (float)(\*net.seen)/N, loss, avg\_loss, get\_current\_rate(net), sec(clock()-time), \*net.seen);

if(\*net.seen/N > epoch){

epoch = \*net.seen/N;

char buff[256];

sprintf(buff, "%s/%s\_%d.weights",backup\_directory,base, epoch);

save\_weights(net, buff);

}

if(get\_current\_batch(net)%100 == 0){

char buff[256];

sprintf(buff, "%s/%s.backup",backup\_directory,base);

save\_weights(net, buff);

}

}

char buff[256];

sprintf(buff, "%s/%s.weights", backup\_directory, base);

save\_weights(net, buff);

free\_network(net);

free\_ptrs((void\*\*)labels, classes);

free\_ptrs((void\*\*)paths, plist->size);

free\_list(plist);

free(base);

}

\*/

void validate\_classifier\_crop(char \*datacfg, char \*filename, char \*weightfile)

{

int i = 0;

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

if(weightfile){

load\_weights(&net, weightfile);

}

srand(time(0));

list \*options = read\_data\_cfg(datacfg);

char \*label\_list = option\_find\_str(options, "labels", "data/labels.list");

char \*valid\_list = option\_find\_str(options, "valid", "data/train.list");

int classes = option\_find\_int(options, "classes", 2);

int topk = option\_find\_int(options, "top", 1);

if (topk > classes) topk = classes;

char \*\*labels = get\_labels(label\_list);

list \*plist = get\_paths(valid\_list);

char \*\*paths = (char \*\*)list\_to\_array(plist);

int m = plist->size;

free\_list(plist);

clock\_t time;

float avg\_acc = 0;

float avg\_topk = 0;

int splits = m/1000;

int num = (i+1)\*m/splits - i\*m/splits;

data val, buffer;

load\_args args = {0};

args.w = net.w;

args.h = net.h;

args.paths = paths;

args.classes = classes;

args.n = num;

args.m = 0;

args.labels = labels;

args.d = &buffer;

args.type = OLD\_CLASSIFICATION\_DATA;

pthread\_t load\_thread = load\_data\_in\_thread(args);

for(i = 1; i <= splits; ++i){

time=clock();

pthread\_join(load\_thread, 0);

val = buffer;

num = (i+1)\*m/splits - i\*m/splits;

char \*\*part = paths+(i\*m/splits);

if(i != splits){

args.paths = part;

load\_thread = load\_data\_in\_thread(args);

}

printf("Loaded: %d images in %lf seconds\n", val.X.rows, sec(clock()-time));

time=clock();

float \*acc = network\_accuracies(net, val, topk);

avg\_acc += acc[0];

avg\_topk += acc[1];

printf("%d: top 1: %f, top %d: %f, %lf seconds, %d images\n", i, avg\_acc/i, topk, avg\_topk/i, sec(clock()-time), val.X.rows);

free\_data(val);

}

}

void validate\_classifier\_10(char \*datacfg, char \*filename, char \*weightfile)

{

int i, j;

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

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

if(weightfile){

load\_weights(&net, weightfile);

}

srand(time(0));

list \*options = read\_data\_cfg(datacfg);

char \*label\_list = option\_find\_str(options, "labels", "data/labels.list");

char \*valid\_list = option\_find\_str(options, "valid", "data/train.list");

int classes = option\_find\_int(options, "classes", 2);

int topk = option\_find\_int(options, "top", 1);

if (topk > classes) topk = classes;

char \*\*labels = get\_labels(label\_list);

list \*plist = get\_paths(valid\_list);

char \*\*paths = (char \*\*)list\_to\_array(plist);

int m = plist->size;

free\_list(plist);

float avg\_acc = 0;

float avg\_topk = 0;

int\* indexes = (int\*)xcalloc(topk, sizeof(int));

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

int class\_id = -1;

char \*path = paths[i];

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

if(strstr(path, labels[j])){

class\_id = j;

break;

}

}

int w = net.w;

int h = net.h;

int shift = 32;

image im = load\_image\_color(paths[i], w+shift, h+shift);

image images[10];

images[0] = crop\_image(im, -shift, -shift, w, h);

images[1] = crop\_image(im, shift, -shift, w, h);

images[2] = crop\_image(im, 0, 0, w, h);

images[3] = crop\_image(im, -shift, shift, w, h);

images[4] = crop\_image(im, shift, shift, w, h);

flip\_image(im);

images[5] = crop\_image(im, -shift, -shift, w, h);

images[6] = crop\_image(im, shift, -shift, w, h);

images[7] = crop\_image(im, 0, 0, w, h);

images[8] = crop\_image(im, -shift, shift, w, h);

images[9] = crop\_image(im, shift, shift, w, h);

float\* pred = (float\*)xcalloc(classes, sizeof(float));

for(j = 0; j < 10; ++j){

float \*p = network\_predict(net, images[j].data);

if(net.hierarchy) hierarchy\_predictions(p, net.outputs, net.hierarchy, 1);

axpy\_cpu(classes, 1, p, 1, pred, 1);

free\_image(images[j]);

}

free\_image(im);

top\_k(pred, classes, topk, indexes);

free(pred);

if(indexes[0] == class\_id) avg\_acc += 1;

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

if(indexes[j] == class\_id) avg\_topk += 1;

}

printf("%d: top 1: %f, top %d: %f\n", i, avg\_acc/(i+1), topk, avg\_topk/(i+1));

}

free(indexes);

}

void validate\_classifier\_full(char \*datacfg, char \*filename, char \*weightfile)

{

int i, j;

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

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

if(weightfile){

load\_weights(&net, weightfile);

}

srand(time(0));

list \*options = read\_data\_cfg(datacfg);

char \*label\_list = option\_find\_str(options, "labels", "data/labels.list");

char \*valid\_list = option\_find\_str(options, "valid", "data/train.list");

int classes = option\_find\_int(options, "classes", 2);

int topk = option\_find\_int(options, "top", 1);

if (topk > classes) topk = classes;

char \*\*labels = get\_labels(label\_list);

list \*plist = get\_paths(valid\_list);

char \*\*paths = (char \*\*)list\_to\_array(plist);

int m = plist->size;

free\_list(plist);

float avg\_acc = 0;

float avg\_topk = 0;

int\* indexes = (int\*)xcalloc(topk, sizeof(int));

int size = net.w;

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

int class\_id = -1;

char \*path = paths[i];

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

if(strstr(path, labels[j])){

class\_id = j;

break;

}

}

image im = load\_image\_color(paths[i], 0, 0);

image resized = resize\_min(im, size);

resize\_network(&net, resized.w, resized.h);

//show\_image(im, "orig");

//show\_image(crop, "cropped");

//cvWaitKey(0);

float \*pred = network\_predict(net, resized.data);

if(net.hierarchy) hierarchy\_predictions(pred, net.outputs, net.hierarchy, 1);

free\_image(im);

free\_image(resized);

top\_k(pred, classes, topk, indexes);

if(indexes[0] == class\_id) avg\_acc += 1;

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

if(indexes[j] == class\_id) avg\_topk += 1;

}

printf("%d: top 1: %f, top %d: %f\n", i, avg\_acc/(i+1), topk, avg\_topk/(i+1));

}

free(indexes);

}

float validate\_classifier\_single(char \*datacfg, char \*filename, char \*weightfile, network \*existing\_net, int topk\_custom)

{

int i, j;

network net;

int old\_batch = -1;

if (existing\_net) {

net = \*existing\_net; // for validation during training

old\_batch = net.batch;

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

}

else {

net = parse\_network\_cfg\_custom(filename, 1, 0);

if (weightfile) {

load\_weights(&net, weightfile);

}

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

fuse\_conv\_batchnorm(net);

calculate\_binary\_weights(net);

}

srand(time(0));

list \*options = read\_data\_cfg(datacfg);

char \*label\_list = option\_find\_str(options, "labels", "data/labels.list");

char \*leaf\_list = option\_find\_str(options, "leaves", 0);

if(leaf\_list) change\_leaves(net.hierarchy, leaf\_list);

char \*valid\_list = option\_find\_str(options, "valid", "data/train.list");

int classes = option\_find\_int(options, "classes", 2);

int topk = option\_find\_int(options, "top", 1);

if (topk\_custom > 0) topk = topk\_custom; // for validation during training

if (topk > classes) topk = classes;

printf(" TOP calculation...\n");

char \*\*labels = get\_labels(label\_list);

list \*plist = get\_paths(valid\_list);

char \*\*paths = (char \*\*)list\_to\_array(plist);

int m = plist->size;

free\_list(plist);

float avg\_acc = 0;

float avg\_topk = 0;

int\* indexes = (int\*)xcalloc(topk, sizeof(int));

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

int class\_id = -1;

char \*path = paths[i];

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

if(strstr(path, labels[j])){

class\_id = j;

break;

}

}

image im = load\_image\_color(paths[i], 0, 0);

image resized = resize\_min(im, net.w);

image crop = crop\_image(resized, (resized.w - net.w)/2, (resized.h - net.h)/2, net.w, net.h);

//show\_image(im, "orig");

//show\_image(crop, "cropped");

//cvWaitKey(0);

float \*pred = network\_predict(net, crop.data);

if(net.hierarchy) hierarchy\_predictions(pred, net.outputs, net.hierarchy, 1);

if(resized.data != im.data) free\_image(resized);

free\_image(im);

free\_image(crop);

top\_k(pred, classes, topk, indexes);

if(indexes[0] == class\_id) avg\_acc += 1;

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

if(indexes[j] == class\_id) avg\_topk += 1;

}

if (existing\_net) printf("\r");

else printf("\n");

printf("%d: top 1: %f, top %d: %f", i, avg\_acc/(i+1), topk, avg\_topk/(i+1));

}

free(indexes);

if (existing\_net) {

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

}

float topk\_result = avg\_topk / i;

return topk\_result;

}

void validate\_classifier\_multi(char \*datacfg, char \*filename, char \*weightfile)

{

int i, j;

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

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

if(weightfile){

load\_weights(&net, weightfile);

}

srand(time(0));

list \*options = read\_data\_cfg(datacfg);

char \*label\_list = option\_find\_str(options, "labels", "data/labels.list");

char \*valid\_list = option\_find\_str(options, "valid", "data/train.list");

int classes = option\_find\_int(options, "classes", 2);

int topk = option\_find\_int(options, "top", 1);

if (topk > classes) topk = classes;

char \*\*labels = get\_labels(label\_list);

list \*plist = get\_paths(valid\_list);

int scales[] = {224, 288, 320, 352, 384};

int nscales = sizeof(scales)/sizeof(scales[0]);

char \*\*paths = (char \*\*)list\_to\_array(plist);

int m = plist->size;

free\_list(plist);

float avg\_acc = 0;

float avg\_topk = 0;

int\* indexes = (int\*)xcalloc(topk, sizeof(int));

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

int class\_id = -1;

char \*path = paths[i];

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

if(strstr(path, labels[j])){

class\_id = j;

break;

}

}

float\* pred = (float\*)xcalloc(classes, sizeof(float));

image im = load\_image\_color(paths[i], 0, 0);

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

image r = resize\_min(im, scales[j]);

resize\_network(&net, r.w, r.h);

float \*p = network\_predict(net, r.data);

if(net.hierarchy) hierarchy\_predictions(p, net.outputs, net.hierarchy, 1);

axpy\_cpu(classes, 1, p, 1, pred, 1);

flip\_image(r);

p = network\_predict(net, r.data);

axpy\_cpu(classes, 1, p, 1, pred, 1);

if(r.data != im.data) free\_image(r);

}

free\_image(im);

top\_k(pred, classes, topk, indexes);

free(pred);

if(indexes[0] == class\_id) avg\_acc += 1;

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

if(indexes[j] == class\_id) avg\_topk += 1;

}

printf("%d: top 1: %f, top %d: %f\n", i, avg\_acc/(i+1), topk, avg\_topk/(i+1));

}

free(indexes);

}

void try\_classifier(char \*datacfg, char \*cfgfile, char \*weightfile, char \*filename, int layer\_num)

{

network net = parse\_network\_cfg\_custom(cfgfile, 1, 0);

if(weightfile){

load\_weights(&net, weightfile);

}

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

srand(2222222);

list \*options = read\_data\_cfg(datacfg);

char \*name\_list = option\_find\_str(options, "names", 0);

if(!name\_list) name\_list = option\_find\_str(options, "labels", "data/labels.list");

int classes = option\_find\_int(options, "classes", 2);

int top = option\_find\_int(options, "top", 1);

if (top > classes) top = classes;

char \*\*names = get\_labels(name\_list);

clock\_t time;

int\* indexes = (int\*)xcalloc(top, sizeof(int));

char buff[256];

char \*input = buff;

while(1){

if(filename){

strncpy(input, filename, 256);

}else{

printf("Enter Image Path: ");

fflush(stdout);

input = fgets(input, 256, stdin);

if(!input) break;

strtok(input, "\n");

}

image orig = load\_image\_color(input, 0, 0);

image r = resize\_min(orig, 256);

image im = crop\_image(r, (r.w - 224 - 1)/2 + 1, (r.h - 224 - 1)/2 + 1, 224, 224);

float mean[] = {0.48263312050943, 0.45230225481413, 0.40099074308742};

float std[] = {0.22590347483426, 0.22120921437787, 0.22103996251583};

float var[3];

var[0] = std[0]\*std[0];

var[1] = std[1]\*std[1];

var[2] = std[2]\*std[2];

normalize\_cpu(im.data, mean, var, 1, 3, im.w\*im.h);

float \*X = im.data;

time=clock();

float \*predictions = network\_predict(net, X);

layer l = net.layers[layer\_num];

int i;

for(i = 0; i < l.c; ++i){

if(l.rolling\_mean) printf("%f %f %f\n", l.rolling\_mean[i], l.rolling\_variance[i], l.scales[i]);

}

#ifdef GPU

cuda\_pull\_array(l.output\_gpu, l.output, l.outputs);

#endif

for(i = 0; i < l.outputs; ++i){

printf("%f\n", l.output[i]);

}

/\*

printf("\n\nWeights\n");

for(i = 0; i < l.n\*l.size\*l.size\*l.c; ++i){

printf("%f\n", l.filters[i]);

}

printf("\n\nBiases\n");

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

printf("%f\n", l.biases[i]);

}

\*/

top\_predictions(net, top, indexes);

printf("%s: Predicted in %f seconds.\n", input, sec(clock()-time));

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

int index = indexes[i];

printf("%s: %f\n", names[index], predictions[index]);

}

free\_image(im);

if (filename) break;

}

free(indexes);

}

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

{

network net = parse\_network\_cfg\_custom(cfgfile, 1, 0);

if(weightfile){

load\_weights(&net, weightfile);

}

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

srand(2222222);

fuse\_conv\_batchnorm(net);

calculate\_binary\_weights(net);

list \*options = read\_data\_cfg(datacfg);

char \*name\_list = option\_find\_str(options, "names", 0);

if(!name\_list) name\_list = option\_find\_str(options, "labels", "data/labels.list");

int classes = option\_find\_int(options, "classes", 2);

printf(" classes = %d, output in cfg = %d \n", classes, net.layers[net.n - 1].c);

if (classes != net.layers[net.n - 1].inputs) {

printf("\n Error: num of filters = %d in the last conv-layer in cfg-file doesn't match to classes = %d in data-file \n",

net.layers[net.n - 1].inputs, classes);

getchar();

}

if (top == 0) top = option\_find\_int(options, "top", 1);

if (top > classes) top = classes;

int i = 0;

char \*\*names = get\_labels(name\_list);

clock\_t time;

int\* indexes = (int\*)xcalloc(top, sizeof(int));

char buff[256];

char \*input = buff;

//int size = net.w;

while(1){

if(filename){

strncpy(input, filename, 256);

}else{

printf("Enter Image Path: ");

fflush(stdout);

input = fgets(input, 256, stdin);

if(!input) break;

strtok(input, "\n");

}

image im = load\_image\_color(input, 0, 0);

image resized = resize\_min(im, net.w);

image r = crop\_image(resized, (resized.w - net.w)/2, (resized.h - net.h)/2, net.w, net.h);

//image r = resize\_min(im, size);

//resize\_network(&net, r.w, r.h);

printf("%d %d\n", r.w, r.h);

float \*X = r.data;

double time = get\_time\_point();

float \*predictions = network\_predict(net, X);

printf("%s: Predicted in %lf milli-seconds.\n", input, ((double)get\_time\_point() - time) / 1000);

if(net.hierarchy) hierarchy\_predictions(predictions, net.outputs, net.hierarchy, 0);

top\_k(predictions, net.outputs, top, indexes);

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

int index = indexes[i];

if(net.hierarchy) printf("%d, %s: %f, parent: %s \n",index, names[index], predictions[index], (net.hierarchy->parent[index] >= 0) ? names[net.hierarchy->parent[index]] : "Root");

else printf("%s: %f\n",names[index], predictions[index]);

}

if(r.data != im.data) free\_image(r);

free\_image(im);

free\_image(resized);

if (filename) break;

}

free(indexes);

free\_network(net);

free\_list\_contents\_kvp(options);

free\_list(options);

}

void label\_classifier(char \*datacfg, char \*filename, char \*weightfile)

{

int i;

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

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

if(weightfile){

load\_weights(&net, weightfile);

}

srand(time(0));

list \*options = read\_data\_cfg(datacfg);

char \*label\_list = option\_find\_str(options, "names", "data/labels.list");

char \*test\_list = option\_find\_str(options, "test", "data/train.list");

int classes = option\_find\_int(options, "classes", 2);

char \*\*labels = get\_labels(label\_list);

list \*plist = get\_paths(test\_list);

char \*\*paths = (char \*\*)list\_to\_array(plist);

int m = plist->size;

free\_list(plist);

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

image im = load\_image\_color(paths[i], 0, 0);

image resized = resize\_min(im, net.w);

image crop = crop\_image(resized, (resized.w - net.w)/2, (resized.h - net.h)/2, net.w, net.h);

float \*pred = network\_predict(net, crop.data);

if(resized.data != im.data) free\_image(resized);

free\_image(im);

free\_image(crop);

int ind = max\_index(pred, classes);

printf("%s\n", labels[ind]);

}

}

void test\_classifier(char \*datacfg, char \*cfgfile, char \*weightfile, int target\_layer)

{

int curr = 0;

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

if(weightfile){

load\_weights(&net, weightfile);

}

srand(time(0));

fuse\_conv\_batchnorm(net);

calculate\_binary\_weights(net);

list \*options = read\_data\_cfg(datacfg);

char \*test\_list = option\_find\_str(options, "test", "data/test.list");

int classes = option\_find\_int(options, "classes", 2);

list \*plist = get\_paths(test\_list);

char \*\*paths = (char \*\*)list\_to\_array(plist);

int m = plist->size;

free\_list(plist);

clock\_t time;

data val, buffer;

load\_args args = {0};

args.w = net.w;

args.h = net.h;

args.paths = paths;

args.classes = classes;

args.n = net.batch;

args.m = 0;

args.labels = 0;

args.d = &buffer;

args.type = OLD\_CLASSIFICATION\_DATA;

pthread\_t load\_thread = load\_data\_in\_thread(args);

for(curr = net.batch; curr < m; curr += net.batch){

time=clock();

pthread\_join(load\_thread, 0);

val = buffer;

if(curr < m){

args.paths = paths + curr;

if (curr + net.batch > m) args.n = m - curr;

load\_thread = load\_data\_in\_thread(args);

}

fprintf(stderr, "Loaded: %d images in %lf seconds\n", val.X.rows, sec(clock()-time));

time=clock();

matrix pred = network\_predict\_data(net, val);

int i, j;

if (target\_layer >= 0){

//layer l = net.layers[target\_layer];

}

for(i = 0; i < pred.rows; ++i){

printf("%s", paths[curr-net.batch+i]);

for(j = 0; j < pred.cols; ++j){

printf("\t%g", pred.vals[i][j]);

}

printf("\n");

}

free\_matrix(pred);

fprintf(stderr, "%lf seconds, %d images, %d total\n", sec(clock()-time), val.X.rows, curr);

free\_data(val);

}

}

void threat\_classifier(char \*datacfg, char \*cfgfile, char \*weightfile, int cam\_index, const char \*filename)

{

#ifdef OPENCV

float threat = 0;

float roll = .2;

printf("Classifier Demo\n");

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

if(weightfile){

load\_weights(&net, weightfile);

}

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

list \*options = read\_data\_cfg(datacfg);

srand(2222222);

cap\_cv \* cap;

if (filename) {

//cap = cvCaptureFromFile(filename);

cap = get\_capture\_video\_stream(filename);

}

else {

//cap = cvCaptureFromCAM(cam\_index);

cap = get\_capture\_webcam(cam\_index);

}

int classes = option\_find\_int(options, "classes", 2);

int top = option\_find\_int(options, "top", 1);

if (top > classes) top = classes;

char \*name\_list = option\_find\_str(options, "names", 0);

char \*\*names = get\_labels(name\_list);

int\* indexes = (int\*)xcalloc(top, sizeof(int));

if(!cap) error("Couldn't connect to webcam.\n");

create\_window\_cv("Threat", 0, 512, 512);

float fps = 0;

int i;

int count = 0;

while(1){

++count;

struct timeval tval\_before, tval\_after, tval\_result;

gettimeofday(&tval\_before, NULL);

//image in = get\_image\_from\_stream(cap);

image in = get\_image\_from\_stream\_cpp(cap);

if(!in.data) break;

image in\_s = resize\_image(in, net.w, net.h);

image out = in;

int x1 = out.w / 20;

int y1 = out.h / 20;

int x2 = 2\*x1;

int y2 = out.h - out.h/20;

int border = .01\*out.h;

int h = y2 - y1 - 2\*border;

int w = x2 - x1 - 2\*border;

float \*predictions = network\_predict(net, in\_s.data);

float curr\_threat = 0;

if(1){

curr\_threat = predictions[0] \* 0 +

predictions[1] \* .6 +

predictions[2];

} else {

curr\_threat = predictions[218] +

predictions[539] +

predictions[540] +

predictions[368] +

predictions[369] +

predictions[370];

}

threat = roll \* curr\_threat + (1-roll) \* threat;

draw\_box\_width(out, x2 + border, y1 + .02\*h, x2 + .5 \* w, y1 + .02\*h + border, border, 0,0,0);

if(threat > .97) {

draw\_box\_width(out, x2 + .5 \* w + border,

y1 + .02\*h - 2\*border,

x2 + .5 \* w + 6\*border,

y1 + .02\*h + 3\*border, 3\*border, 1,0,0);

}

draw\_box\_width(out, x2 + .5 \* w + border,

y1 + .02\*h - 2\*border,

x2 + .5 \* w + 6\*border,

y1 + .02\*h + 3\*border, .5\*border, 0,0,0);

draw\_box\_width(out, x2 + border, y1 + .42\*h, x2 + .5 \* w, y1 + .42\*h + border, border, 0,0,0);

if(threat > .57) {

draw\_box\_width(out, x2 + .5 \* w + border,

y1 + .42\*h - 2\*border,

x2 + .5 \* w + 6\*border,

y1 + .42\*h + 3\*border, 3\*border, 1,1,0);

}

draw\_box\_width(out, x2 + .5 \* w + border,

y1 + .42\*h - 2\*border,

x2 + .5 \* w + 6\*border,

y1 + .42\*h + 3\*border, .5\*border, 0,0,0);

draw\_box\_width(out, x1, y1, x2, y2, border, 0,0,0);

for(i = 0; i < threat \* h ; ++i){

float ratio = (float) i / h;

float r = (ratio < .5) ? (2\*(ratio)) : 1;

float g = (ratio < .5) ? 1 : 1 - 2\*(ratio - .5);

draw\_box\_width(out, x1 + border, y2 - border - i, x2 - border, y2 - border - i, 1, r, g, 0);

}

top\_predictions(net, top, indexes);

char buff[256];

sprintf(buff, "tmp/threat\_%06d", count);

//save\_image(out, buff);

#ifndef \_WIN32

printf("\033[2J");

printf("\033[1;1H");

#endif

printf("\nFPS:%.0f\n",fps);

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

int index = indexes[i];

printf("%.1f%%: %s\n", predictions[index]\*100, names[index]);

}

if(1){

show\_image(out, "Threat");

wait\_key\_cv(10);

}

free\_image(in\_s);

free\_image(in);

gettimeofday(&tval\_after, NULL);

timersub(&tval\_after, &tval\_before, &tval\_result);

float curr = 1000000.f/((long int)tval\_result.tv\_usec);

fps = .9\*fps + .1\*curr;

}

#endif

}

void gun\_classifier(char \*datacfg, char \*cfgfile, char \*weightfile, int cam\_index, const char \*filename)

{

#ifdef OPENCV\_DISABLE

int bad\_cats[] = {218, 539, 540, 1213, 1501, 1742, 1911, 2415, 4348, 19223, 368, 369, 370, 1133, 1200, 1306, 2122, 2301, 2537, 2823, 3179, 3596, 3639, 4489, 5107, 5140, 5289, 6240, 6631, 6762, 7048, 7171, 7969, 7984, 7989, 8824, 8927, 9915, 10270, 10448, 13401, 15205, 18358, 18894, 18895, 19249, 19697};

printf("Classifier Demo\n");

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

if(weightfile){

load\_weights(&net, weightfile);

}

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

list \*options = read\_data\_cfg(datacfg);

srand(2222222);

CvCapture \* cap;

if (filename) {

//cap = cvCaptureFromFile(filename);

cap = get\_capture\_video\_stream(filename);

}

else {

//cap = cvCaptureFromCAM(cam\_index);

cap = get\_capture\_webcam(cam\_index);

}

int classes = option\_find\_int(options, "classes", 2);

int top = option\_find\_int(options, "top", 1);

if (top > classes) top = classes;

char \*name\_list = option\_find\_str(options, "names", 0);

char \*\*names = get\_labels(name\_list);

int\* indexes = (int\*)xcalloc(top, sizeof(int));

if(!cap) error("Couldn't connect to webcam.\n");

cvNamedWindow("Threat Detection", CV\_WINDOW\_NORMAL);

cvResizeWindow("Threat Detection", 512, 512);

float fps = 0;

int i;

while(1){

struct timeval tval\_before, tval\_after, tval\_result;

gettimeofday(&tval\_before, NULL);

//image in = get\_image\_from\_stream(cap);

image in = get\_image\_from\_stream\_cpp(cap);

image in\_s = resize\_image(in, net.w, net.h);

show\_image(in, "Threat Detection");

float \*predictions = network\_predict(net, in\_s.data);

top\_predictions(net, top, indexes);

printf("\033[2J");

printf("\033[1;1H");

int threat = 0;

for(i = 0; i < sizeof(bad\_cats)/sizeof(bad\_cats[0]); ++i){

int index = bad\_cats[i];

if(predictions[index] > .01){

printf("Threat Detected!\n");

threat = 1;

break;

}

}

if(!threat) printf("Scanning...\n");

for(i = 0; i < sizeof(bad\_cats)/sizeof(bad\_cats[0]); ++i){

int index = bad\_cats[i];

if(predictions[index] > .01){

printf("%s\n", names[index]);

}

}

free\_image(in\_s);

free\_image(in);

cvWaitKey(10);

gettimeofday(&tval\_after, NULL);

timersub(&tval\_after, &tval\_before, &tval\_result);

float curr = 1000000.f/((long int)tval\_result.tv\_usec);

fps = .9\*fps + .1\*curr;

}

#endif

}

void demo\_classifier(char \*datacfg, char \*cfgfile, char \*weightfile, int cam\_index, const char \*filename, int benchmark, int benchmark\_layers)

{

#ifdef OPENCV

printf("Classifier Demo\n");

network net = parse\_network\_cfg\_custom(cfgfile, 1, 0);

if(weightfile){

load\_weights(&net, weightfile);

}

net.benchmark\_layers = benchmark\_layers;

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

list \*options = read\_data\_cfg(datacfg);

fuse\_conv\_batchnorm(net);

calculate\_binary\_weights(net);

srand(2222222);

cap\_cv \* cap;

if(filename){

cap = get\_capture\_video\_stream(filename);

}else{

cap = get\_capture\_webcam(cam\_index);

}

int classes = option\_find\_int(options, "classes", 2);

int top = option\_find\_int(options, "top", 1);

if (top > classes) top = classes;

char \*name\_list = option\_find\_str(options, "names", 0);

char \*\*names = get\_labels(name\_list);

int\* indexes = (int\*)xcalloc(top, sizeof(int));

if(!cap) error("Couldn't connect to webcam.\n");

if (!benchmark) create\_window\_cv("Classifier", 0, 512, 512);

float fps = 0;

int i;

double start\_time = get\_time\_point();

float avg\_fps = 0;

int frame\_counter = 0;

while(1){

struct timeval tval\_before, tval\_after, tval\_result;

gettimeofday(&tval\_before, NULL);

//image in = get\_image\_from\_stream(cap);

image in\_s, in;

if (!benchmark) {

in = get\_image\_from\_stream\_cpp(cap);

in\_s = resize\_image(in, net.w, net.h);

show\_image(in, "Classifier");

}

else {

static image tmp;

if (!tmp.data) tmp = make\_image(net.w, net.h, 3);

in\_s = tmp;

}

double time = get\_time\_point();

float \*predictions = network\_predict(net, in\_s.data);

double frame\_time\_ms = (get\_time\_point() - time)/1000;

frame\_counter++;

if(net.hierarchy) hierarchy\_predictions(predictions, net.outputs, net.hierarchy, 1);

top\_predictions(net, top, indexes);

#ifndef \_WIN32

printf("\033[2J");

printf("\033[1;1H");

#endif

if (!benchmark) {

printf("\rFPS: %.2f (use -benchmark command line flag for correct measurement)\n", fps);

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

int index = indexes[i];

printf("%.1f%%: %s\n", predictions[index] \* 100, names[index]);

}

printf("\n");

free\_image(in\_s);

free\_image(in);

int c = wait\_key\_cv(10);// cvWaitKey(10);

if (c == 27 || c == 1048603) break;

}

else {

printf("\rFPS: %.2f \t AVG\_FPS = %.2f ", fps, avg\_fps);

}

//gettimeofday(&tval\_after, NULL);

//timersub(&tval\_after, &tval\_before, &tval\_result);

//float curr = 1000000.f/((long int)tval\_result.tv\_usec);

float curr = 1000.f / frame\_time\_ms;

if (fps == 0) fps = curr;

else fps = .9\*fps + .1\*curr;

float spent\_time = (get\_time\_point() - start\_time) / 1000000;

if (spent\_time >= 3.0f) {

//printf(" spent\_time = %f \n", spent\_time);

avg\_fps = frame\_counter / spent\_time;

frame\_counter = 0;

start\_time = get\_time\_point();

}

}

#endif

}

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

{

if(argc < 4){

fprintf(stderr, "usage: %s %s [train/test/valid] [cfg] [weights (optional)]\n", argv[0], argv[1]);

return;

}

int mjpeg\_port = find\_int\_arg(argc, argv, "-mjpeg\_port", -1);

char \*gpu\_list = find\_char\_arg(argc, argv, "-gpus", 0);

int \*gpus = 0;

int gpu = 0;

int ngpus = 0;

if(gpu\_list){

printf("%s\n", gpu\_list);

int len = strlen(gpu\_list);

ngpus = 1;

int i;

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

if (gpu\_list[i] == ',') ++ngpus;

}

gpus = (int\*)xcalloc(ngpus, sizeof(int));

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

gpus[i] = atoi(gpu\_list);

gpu\_list = strchr(gpu\_list, ',')+1;

}

} else {

gpu = gpu\_index;

gpus = &gpu;

ngpus = 1;

}

int dont\_show = find\_arg(argc, argv, "-dont\_show");

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

int benchmark\_layers = find\_arg(argc, argv, "-benchmark\_layers");

if (benchmark\_layers) benchmark = 1;

int dontuse\_opencv = find\_arg(argc, argv, "-dontuse\_opencv");

int show\_imgs = find\_arg(argc, argv, "-show\_imgs");

int calc\_topk = find\_arg(argc, argv, "-topk");

int cam\_index = find\_int\_arg(argc, argv, "-c", 0);

int top = find\_int\_arg(argc, argv, "-t", 0);

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

char \*data = argv[3];

char \*cfg = argv[4];

char \*weights = (argc > 5) ? argv[5] : 0;

char \*filename = (argc > 6) ? argv[6]: 0;

char \*layer\_s = (argc > 7) ? argv[7]: 0;

int layer = layer\_s ? atoi(layer\_s) : -1;

char\* chart\_path = find\_char\_arg(argc, argv, "-chart", 0);

if(0==strcmp(argv[2], "predict")) predict\_classifier(data, cfg, weights, filename, top);

else if(0==strcmp(argv[2], "try")) try\_classifier(data, cfg, weights, filename, atoi(layer\_s));

else if(0==strcmp(argv[2], "train")) train\_classifier(data, cfg, weights, gpus, ngpus, clear, dontuse\_opencv, dont\_show, mjpeg\_port, calc\_topk, show\_imgs, chart\_path);

else if(0==strcmp(argv[2], "demo")) demo\_classifier(data, cfg, weights, cam\_index, filename, benchmark, benchmark\_layers);

else if(0==strcmp(argv[2], "gun")) gun\_classifier(data, cfg, weights, cam\_index, filename);

else if(0==strcmp(argv[2], "threat")) threat\_classifier(data, cfg, weights, cam\_index, filename);

else if(0==strcmp(argv[2], "test")) test\_classifier(data, cfg, weights, layer);

else if(0==strcmp(argv[2], "label")) label\_classifier(data, cfg, weights);

else if(0==strcmp(argv[2], "valid")) validate\_classifier\_single(data, cfg, weights, NULL, -1);

else if(0==strcmp(argv[2], "validmulti")) validate\_classifier\_multi(data, cfg, weights);

else if(0==strcmp(argv[2], "valid10")) validate\_classifier\_10(data, cfg, weights);

else if(0==strcmp(argv[2], "validcrop")) validate\_classifier\_crop(data, cfg, weights);

else if(0==strcmp(argv[2], "validfull")) validate\_classifier\_full(data, cfg, weights);

if (gpus && gpu\_list && ngpus > 1) free(gpus);

}