#include "network.h"

#include "utils.h"

#include "parser.h"

void fix\_data\_captcha(data d, int mask)

{

matrix labels = d.y;

int i, j;

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

for(j = 0; j < d.y.cols; j += 2){

if (mask){

if(!labels.vals[i][j]){

labels.vals[i][j] = SECRET\_NUM;

labels.vals[i][j+1] = SECRET\_NUM;

}else if(labels.vals[i][j+1]){

labels.vals[i][j] = 0;

}

} else{

if (labels.vals[i][j]) {

labels.vals[i][j+1] = 0;

} else {

labels.vals[i][j+1] = 1;

}

}

}

}

}

void train\_captcha(char \*cfgfile, char \*weightfile)

{

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);

}

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

int imgs = 1024;

int i = \*net.seen/imgs;

int solved = 1;

list \*plist;

char\*\* labels = get\_labels("data/captcha/reimgs.labels.list");

if (solved){

plist = get\_paths("data/captcha/reimgs.solved.list");

}else{

plist = get\_paths("data/captcha/reimgs.raw.list");

}

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

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

clock\_t time;

pthread\_t load\_thread;

data train;

data buffer;

load\_args args = {0};

args.w = net.w;

args.h = net.h;

args.paths = paths;

args.classes = 26;

args.n = imgs;

args.m = plist->size;

args.labels = labels;

args.d = &buffer;

args.type = CLASSIFICATION\_DATA;

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

while(1){

++i;

time=clock();

pthread\_join(load\_thread, 0);

train = buffer;

fix\_data\_captcha(train, solved);

/\*

image im = float\_to\_image(256, 256, 3, train.X.vals[114]);

show\_image(im, "training");

cvWaitKey(0);

\*/

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

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

time=clock();

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

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

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

printf("%d: %f, %f avg, %lf seconds, %ld images\n", i, loss, avg\_loss, sec(clock()-time), \*net.seen);

free\_data(train);

if(i%100==0){

char buff[256];

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

save\_weights(net, buff);

}

}

}

void test\_captcha(char \*cfgfile, char \*weightfile, char \*filename)

{

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

if(weightfile){

load\_weights(&net, weightfile);

}

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

srand(2222222);

int i = 0;

char\*\* names = get\_labels("data/captcha/reimgs.labels.list");

char buff[256];

char \*input = buff;

int indexes[26];

while(1){

if(filename){

strncpy(input, filename, 256);

}else{

//printf("Enter Image Path: ");

//fflush(stdout);

input = fgets(input, 256, stdin);

if(!input) return;

strtok(input, "\n");

}

image im = load\_image\_color(input, net.w, net.h);

float \*X = im.data;

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

top\_predictions(net, 26, indexes);

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

for(i = 0; i < 26; ++i){

int index = indexes[i];

if(i != 0) printf(", ");

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

}

printf("\n");

fflush(stdout);

free\_image(im);

if (filename) break;

}

}

void valid\_captcha(char \*cfgfile, char \*weightfile, char \*filename)

{

char\*\* labels = get\_labels("data/captcha/reimgs.labels.list");

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

if(weightfile){

load\_weights(&net, weightfile);

}

list\* plist = get\_paths("data/captcha/reimgs.fg.list");

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

int N = plist->size;

int outputs = net.outputs;

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

srand(2222222);

int i, j;

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

if (i%100 == 0) fprintf(stderr, "%d\n", i);

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

float \*X = im.data;

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

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

int truth = -1;

for(j = 0; j < 13; ++j){

if (strstr(paths[i], labels[j])) truth = j;

}

if (truth == -1){

fprintf(stderr, "bad: %s\n", paths[i]);

return;

}

printf("%d, ", truth);

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

if (j != 0) printf(", ");

printf("%f", predictions[j]);

}

printf("\n");

fflush(stdout);

free\_image(im);

if (filename) break;

}

}

/\*

void train\_captcha(char \*cfgfile, char \*weightfile)

{

float avg\_loss = -1;

srand(time(0));

char \*base = basecfg(cfgfile);

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

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

if(weightfile){

load\_weights(&net, weightfile);

}

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

int imgs = 1024;

int i = net.seen/imgs;

list \*plist = get\_paths("/data/captcha/train.auto5");

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

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

clock\_t time;

while(1){

++i;

time=clock();

data train = load\_data\_captcha(paths, imgs, plist->size, 10, 200, 60);

translate\_data\_rows(train, -128);

scale\_data\_rows(train, 1./128);

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

time=clock();

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

net.seen += imgs;

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

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

printf("%d: %f, %f avg, %lf seconds, %d images\n", i, loss, avg\_loss, sec(clock()-time), net.seen);

free\_data(train);

if(i%10==0){

char buff[256];

sprintf(buff, "/home/pjreddie/imagenet\_backup/%s\_%d.weights",base, i);

save\_weights(net, buff);

}

}

}

void decode\_captcha(char \*cfgfile, char \*weightfile)

{

setbuf(stdout, NULL);

srand(time(0));

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

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

if(weightfile){

load\_weights(&net, weightfile);

}

char filename[256];

while(1){

printf("Enter filename: ");

fgets(filename, 256, stdin);

strtok(filename, "\n");

image im = load\_image\_color(filename, 300, 57);

scale\_image(im, 1./255.);

float \*X = im.data;

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

image out = float\_to\_image(300, 57, 1, predictions);

show\_image(out, "decoded");

#ifdef OPENCV

cvWaitKey(0);

#endif

free\_image(im);

}

}

void encode\_captcha(char \*cfgfile, char \*weightfile)

{

float avg\_loss = -1;

srand(time(0));

char \*base = basecfg(cfgfile);

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

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

if(weightfile){

load\_weights(&net, weightfile);

}

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

int imgs = 1024;

int i = net.seen/imgs;

list \*plist = get\_paths("/data/captcha/encode.list");

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

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

clock\_t time;

while(1){

++i;

time=clock();

data train = load\_data\_captcha\_encode(paths, imgs, plist->size, 300, 57);

scale\_data\_rows(train, 1./255);

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

time=clock();

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

net.seen += imgs;

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

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

printf("%d: %f, %f avg, %lf seconds, %d images\n", i, loss, avg\_loss, sec(clock()-time), net.seen);

free\_matrix(train.X);

if(i%100==0){

char buff[256];

sprintf(buff, "/home/pjreddie/imagenet\_backup/%s\_%d.weights",base, i);

save\_weights(net, buff);

}

}

}

void validate\_captcha(char \*cfgfile, char \*weightfile)

{

srand(time(0));

char \*base = basecfg(cfgfile);

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

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

if(weightfile){

load\_weights(&net, weightfile);

}

int numchars = 37;

list \*plist = get\_paths("/data/captcha/solved.hard");

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

int imgs = plist->size;

data valid = load\_data\_captcha(paths, imgs, 0, 10, 200, 60);

translate\_data\_rows(valid, -128);

scale\_data\_rows(valid, 1./128);

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

int i, k;

int correct = 0;

int total = 0;

int accuracy = 0;

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

int allcorrect = 1;

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

char truth = int\_to\_alphanum(max\_index(valid.y.vals[i]+k\*numchars, numchars));

char prediction = int\_to\_alphanum(max\_index(pred.vals[i]+k\*numchars, numchars));

if (truth != prediction) allcorrect=0;

if (truth != '.' && truth == prediction) ++correct;

if (truth != '.' || truth != prediction) ++total;

}

accuracy += allcorrect;

}

printf("Word Accuracy: %f, Char Accuracy %f\n", (float)accuracy/imgs, (float)correct/total);

free\_data(valid);

}

void test\_captcha(char \*cfgfile, char \*weightfile)

{

setbuf(stdout, NULL);

srand(time(0));

//char \*base = basecfg(cfgfile);

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

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

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

if(weightfile){

load\_weights(&net, weightfile);

}

char filename[256];

while(1){

//printf("Enter filename: ");

fgets(filename, 256, stdin);

strtok(filename, "\n");

image im = load\_image\_color(filename, 200, 60);

translate\_image(im, -128);

scale\_image(im, 1/128.);

float \*X = im.data;

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

print\_letters(predictions, 10);

free\_image(im);

}

}

\*/

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

{

if(argc < 4){

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

return;

}

char \*cfg = argv[3];

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

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

if(0==strcmp(argv[2], "train")) train\_captcha(cfg, weights);

else if(0==strcmp(argv[2], "test")) test\_captcha(cfg, weights, filename);

else if(0==strcmp(argv[2], "valid")) valid\_captcha(cfg, weights, filename);

//if(0==strcmp(argv[2], "test")) test\_captcha(cfg, weights);

//else if(0==strcmp(argv[2], "encode")) encode\_captcha(cfg, weights);

//else if(0==strcmp(argv[2], "decode")) decode\_captcha(cfg, weights);

//else if(0==strcmp(argv[2], "valid")) validate\_captcha(cfg, weights);

}