#include "detection\_layer.h"

#include "activations.h"

#include "softmax\_layer.h"

#include "blas.h"

#include "box.h"

#include "dark\_cuda.h"

#include "utils.h"

#include <stdio.h>

#include <assert.h>

#include <string.h>

#include <stdlib.h>

detection\_layer make\_detection\_layer(int batch, int inputs, int n, int side, int classes, int coords, int rescore)

{

detection\_layer l = { (LAYER\_TYPE)0 };

l.type = DETECTION;

l.n = n;

l.batch = batch;

l.inputs = inputs;

l.classes = classes;

l.coords = coords;

l.rescore = rescore;

l.side = side;

l.w = side;

l.h = side;

assert(side\*side\*((1 + l.coords)\*l.n + l.classes) == inputs);

l.cost = (float\*)xcalloc(1, sizeof(float));

l.outputs = l.inputs;

l.truths = l.side\*l.side\*(1+l.coords+l.classes);

l.output = (float\*)xcalloc(batch \* l.outputs, sizeof(float));

l.delta = (float\*)xcalloc(batch \* l.outputs, sizeof(float));

l.forward = forward\_detection\_layer;

l.backward = backward\_detection\_layer;

#ifdef GPU

l.forward\_gpu = forward\_detection\_layer\_gpu;

l.backward\_gpu = backward\_detection\_layer\_gpu;

l.output\_gpu = cuda\_make\_array(l.output, batch\*l.outputs);

l.delta\_gpu = cuda\_make\_array(l.delta, batch\*l.outputs);

#endif

fprintf(stderr, "Detection Layer\n");

srand(time(0));

return l;

}

void forward\_detection\_layer(const detection\_layer l, network\_state state)

{

int locations = l.side\*l.side;

int i,j;

memcpy(l.output, state.input, l.outputs\*l.batch\*sizeof(float));

//if(l.reorg) reorg(l.output, l.w\*l.h, size\*l.n, l.batch, 1);

int b;

if (l.softmax){

for(b = 0; b < l.batch; ++b){

int index = b\*l.inputs;

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

int offset = i\*l.classes;

softmax(l.output + index + offset, l.classes, 1,

l.output + index + offset, 1);

}

}

}

if(state.train){

float avg\_iou = 0;

float avg\_cat = 0;

float avg\_allcat = 0;

float avg\_obj = 0;

float avg\_anyobj = 0;

int count = 0;

\*(l.cost) = 0;

int size = l.inputs \* l.batch;

memset(l.delta, 0, size \* sizeof(float));

for (b = 0; b < l.batch; ++b){

int index = b\*l.inputs;

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

int truth\_index = (b\*locations + i)\*(1+l.coords+l.classes);

int is\_obj = state.truth[truth\_index];

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

int p\_index = index + locations\*l.classes + i\*l.n + j;

l.delta[p\_index] = l.noobject\_scale\*(0 - l.output[p\_index]);

\*(l.cost) += l.noobject\_scale\*pow(l.output[p\_index], 2);

avg\_anyobj += l.output[p\_index];

}

int best\_index = -1;

float best\_iou = 0;

float best\_rmse = 20;

if (!is\_obj){

continue;

}

int class\_index = index + i\*l.classes;

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

l.delta[class\_index+j] = l.class\_scale \* (state.truth[truth\_index+1+j] - l.output[class\_index+j]);

\*(l.cost) += l.class\_scale \* pow(state.truth[truth\_index+1+j] - l.output[class\_index+j], 2);

if(state.truth[truth\_index + 1 + j]) avg\_cat += l.output[class\_index+j];

avg\_allcat += l.output[class\_index+j];

}

box truth = float\_to\_box(state.truth + truth\_index + 1 + l.classes);

truth.x /= l.side;

truth.y /= l.side;

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

int box\_index = index + locations\*(l.classes + l.n) + (i\*l.n + j) \* l.coords;

box out = float\_to\_box(l.output + box\_index);

out.x /= l.side;

out.y /= l.side;

if (l.sqrt){

out.w = out.w\*out.w;

out.h = out.h\*out.h;

}

float iou = box\_iou(out, truth);

//iou = 0;

float rmse = box\_rmse(out, truth);

if(best\_iou > 0 || iou > 0){

if(iou > best\_iou){

best\_iou = iou;

best\_index = j;

}

}else{

if(rmse < best\_rmse){

best\_rmse = rmse;

best\_index = j;

}

}

}

if(l.forced){

if(truth.w\*truth.h < .1){

best\_index = 1;

}else{

best\_index = 0;

}

}

if(l.random && \*(state.net.seen) < 64000){

best\_index = rand()%l.n;

}

int box\_index = index + locations\*(l.classes + l.n) + (i\*l.n + best\_index) \* l.coords;

int tbox\_index = truth\_index + 1 + l.classes;

box out = float\_to\_box(l.output + box\_index);

out.x /= l.side;

out.y /= l.side;

if (l.sqrt) {

out.w = out.w\*out.w;

out.h = out.h\*out.h;

}

float iou = box\_iou(out, truth);

//printf("%d,", best\_index);

int p\_index = index + locations\*l.classes + i\*l.n + best\_index;

\*(l.cost) -= l.noobject\_scale \* pow(l.output[p\_index], 2);

\*(l.cost) += l.object\_scale \* pow(1-l.output[p\_index], 2);

avg\_obj += l.output[p\_index];

l.delta[p\_index] = l.object\_scale \* (1.-l.output[p\_index]);

if(l.rescore){

l.delta[p\_index] = l.object\_scale \* (iou - l.output[p\_index]);

}

l.delta[box\_index+0] = l.coord\_scale\*(state.truth[tbox\_index + 0] - l.output[box\_index + 0]);

l.delta[box\_index+1] = l.coord\_scale\*(state.truth[tbox\_index + 1] - l.output[box\_index + 1]);

l.delta[box\_index+2] = l.coord\_scale\*(state.truth[tbox\_index + 2] - l.output[box\_index + 2]);

l.delta[box\_index+3] = l.coord\_scale\*(state.truth[tbox\_index + 3] - l.output[box\_index + 3]);

if(l.sqrt){

l.delta[box\_index+2] = l.coord\_scale\*(sqrt(state.truth[tbox\_index + 2]) - l.output[box\_index + 2]);

l.delta[box\_index+3] = l.coord\_scale\*(sqrt(state.truth[tbox\_index + 3]) - l.output[box\_index + 3]);

}

\*(l.cost) += pow(1-iou, 2);

avg\_iou += iou;

++count;

}

}

if(0){

float\* costs = (float\*)xcalloc(l.batch \* locations \* l.n, sizeof(float));

for (b = 0; b < l.batch; ++b) {

int index = b\*l.inputs;

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

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

int p\_index = index + locations\*l.classes + i\*l.n + j;

costs[b\*locations\*l.n + i\*l.n + j] = l.delta[p\_index]\*l.delta[p\_index];

}

}

}

int indexes[100];

top\_k(costs, l.batch\*locations\*l.n, 100, indexes);

float cutoff = costs[indexes[99]];

for (b = 0; b < l.batch; ++b) {

int index = b\*l.inputs;

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

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

int p\_index = index + locations\*l.classes + i\*l.n + j;

if (l.delta[p\_index]\*l.delta[p\_index] < cutoff) l.delta[p\_index] = 0;

}

}

}

free(costs);

}

\*(l.cost) = pow(mag\_array(l.delta, l.outputs \* l.batch), 2);

printf("Detection Avg IOU: %f, Pos Cat: %f, All Cat: %f, Pos Obj: %f, Any Obj: %f, count: %d\n", avg\_iou/count, avg\_cat/count, avg\_allcat/(count\*l.classes), avg\_obj/count, avg\_anyobj/(l.batch\*locations\*l.n), count);

//if(l.reorg) reorg(l.delta, l.w\*l.h, size\*l.n, l.batch, 0);

}

}

void backward\_detection\_layer(const detection\_layer l, network\_state state)

{

axpy\_cpu(l.batch\*l.inputs, 1, l.delta, 1, state.delta, 1);

}

void get\_detection\_boxes(layer l, int w, int h, float thresh, float \*\*probs, box \*boxes, int only\_objectness)

{

int i,j,n;

float \*predictions = l.output;

//int per\_cell = 5\*num+classes;

for (i = 0; i < l.side\*l.side; ++i){

int row = i / l.side;

int col = i % l.side;

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

int index = i\*l.n + n;

int p\_index = l.side\*l.side\*l.classes + i\*l.n + n;

float scale = predictions[p\_index];

int box\_index = l.side\*l.side\*(l.classes + l.n) + (i\*l.n + n)\*4;

boxes[index].x = (predictions[box\_index + 0] + col) / l.side \* w;

boxes[index].y = (predictions[box\_index + 1] + row) / l.side \* h;

boxes[index].w = pow(predictions[box\_index + 2], (l.sqrt?2:1)) \* w;

boxes[index].h = pow(predictions[box\_index + 3], (l.sqrt?2:1)) \* h;

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

int class\_index = i\*l.classes;

float prob = scale\*predictions[class\_index+j];

probs[index][j] = (prob > thresh) ? prob : 0;

}

if(only\_objectness){

probs[index][0] = scale;

}

}

}

}

#ifdef GPU

void forward\_detection\_layer\_gpu(const detection\_layer l, network\_state state)

{

if(!state.train){

copy\_ongpu(l.batch\*l.inputs, state.input, 1, l.output\_gpu, 1);

return;

}

float\* in\_cpu = (float\*)xcalloc(l.batch \* l.inputs, sizeof(float));

float \*truth\_cpu = 0;

if(state.truth){

int num\_truth = l.batch\*l.side\*l.side\*(1+l.coords+l.classes);

truth\_cpu = (float\*)xcalloc(num\_truth, sizeof(float));

cuda\_pull\_array(state.truth, truth\_cpu, num\_truth);

}

cuda\_pull\_array(state.input, in\_cpu, l.batch\*l.inputs);

network\_state cpu\_state = state;

cpu\_state.train = state.train;

cpu\_state.truth = truth\_cpu;

cpu\_state.input = in\_cpu;

forward\_detection\_layer(l, cpu\_state);

cuda\_push\_array(l.output\_gpu, l.output, l.batch\*l.outputs);

cuda\_push\_array(l.delta\_gpu, l.delta, l.batch\*l.inputs);

free(cpu\_state.input);

if(cpu\_state.truth) free(cpu\_state.truth);

}

void backward\_detection\_layer\_gpu(detection\_layer l, network\_state state)

{

axpy\_ongpu(l.batch\*l.inputs, 1, l.delta\_gpu, 1, state.delta, 1);

//copy\_ongpu(l.batch\*l.inputs, l.delta\_gpu, 1, state.delta, 1);

}

#endif

void get\_detection\_detections(layer l, int w, int h, float thresh, detection \*dets)

{

int i, j, n;

float \*predictions = l.output;

//int per\_cell = 5\*num+classes;

for (i = 0; i < l.side\*l.side; ++i) {

int row = i / l.side;

int col = i % l.side;

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

int index = i\*l.n + n;

int p\_index = l.side\*l.side\*l.classes + i\*l.n + n;

float scale = predictions[p\_index];

int box\_index = l.side\*l.side\*(l.classes + l.n) + (i\*l.n + n) \* 4;

box b;

b.x = (predictions[box\_index + 0] + col) / l.side \* w;

b.y = (predictions[box\_index + 1] + row) / l.side \* h;

b.w = pow(predictions[box\_index + 2], (l.sqrt ? 2 : 1)) \* w;

b.h = pow(predictions[box\_index + 3], (l.sqrt ? 2 : 1)) \* h;

dets[index].bbox = b;

dets[index].objectness = scale;

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

int class\_index = i\*l.classes;

float prob = scale\*predictions[class\_index + j];

dets[index].prob[j] = (prob > thresh) ? prob : 0;

}

}

}

}