**[regression]**

**output=1**

**w=1**

**h=1**

**调试：**

**rotate train C:\Users\zheng\Desktop\GPU\_darknet-master\darknet-master\build\darknet\x64\rotate\rotate.data C:\Users\zheng\Desktop\GPU\_darknet-master\darknet-master\build\darknet\x64\rotate\small\_.cfg**

**classifier train C:\Users\zheng\Desktop\GPU\_darknet-master\darknet-master\build\darknet\x64\classify\classify.data C:\Users\zheng\Desktop\GPU\_darknet-master\darknet-master\build\darknet\x64\classify\small\_.cfg**

**rotate test C:\Users\zheng\Desktop\darknet-master\build\darknet\x64\rotate\rotate.data C:\Users\zheng\Desktop\darknet-master\build\darknet\x64\rotate\small.cfg C:\Users\zheng\Desktop\darknet-master\build\darknet\x64\rotate\backup\small\_last.weights**

**Darknet.c中添加：**

extern void run\_rotate(int argc, char\*\* argv);//rotate

else if (0 == strcmp(argv[1], "rotate")) { //rotate

run\_rotate(argc, argv);

}

**darknet.h里添加：**

data\_type;

ROTATE\_DATA

typedef enum

REGRESSION,//rotate

// layer.h

typedef enum{

SSE, MASKED, L1, SEG, SMOOTH,WGAN,ROTATE//rotate

} COST\_TYPE;

// data.h

char\*\* rotate\_labels;//rotate

LIB\_API image load\_image\_gray(char\* filename, int w, int h);//rotate

**Parser.c中添加：**

#include "regression\_layer.h"//rotate

string\_to\_layer\_type函数中添加

if (strcmp(type, "[regression]") == 0) return REGRESSION; //rotate

parse\_network\_cfg中加入

else if (lt == REGRESSION){//rotate

l = parse\_ regression(options, params);

}

添加parse\_ regression函数

layer parse\_regression(list\* options, size\_params params)//rotate

{

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

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

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

layer l = make\_regression\_layer(params.batch, params.inputs, output, w, h);

return l;

} }

**network.c中添加：**

#include "regression \_layer.h"//rotate

get\_layer\_string函数中添加

case REGRESSION:

return "regression";//rotate

resize\_network函数中添加：

//else if (l.type == REGRESSION){//线性回归

// resize\_regression(&l, w, h);//..

//}

**gemm.c中修改：(暂不修改)**

void gemm(int TA, int TB, int M, int N, int K, float ALPHA,

float \*A, int lda,

float \*B, int ldb,

float BETA,

float \*C, int ldc)

gemm\_cpu( TA, TB, M, N, K, ALPHA/ALPHA, A,lda, B, ldb, BETA/BETA, C,ldc); //ALPHA BETA

**data.c中添加：**

else if (a.type == ROTATE\_DATA) {//rotate

\*a.d = load\_data\_rotate(a.paths, a.n, a.m, a.rotate\_train\_labels, 1, a.hierarchy, a.flip, a.min, a.max, a.w, a.h, a.angle, a.aspect, a.hue, a.saturation, a.exposure, a.mixup, a.blur, a.show\_imgs, a.label\_smooth\_eps, a.dontuse\_opencv, a.contrastive);

} }

//rotate

void fill\_truth\_rotate(char\*\* paths, int n, float\* truth, float\* labels) {

int j;

int idx;

if (labels) {

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

char\* p = *strrchr*(paths[j], '\\');

idx = *atoi*(p + 1);

truth[j] = labels[idx];

}

}

}

char\*\* get\_rotate\_labels\_custom(char\* filename, int\* size)

{

FILE\* fp = fopen("D:/string.txt", "wb");

char ch='a';

if (fp ) {

fwrite(&ch, 1, 1, fp);

fclose(fp);

}

list\* plist = get\_paths(filename);

if (size) \*size = plist->size;

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

free\_list(plist);

return labels;

}//..

char \*\*get\_rotate\_labels(char\* filename, int\* size)

{

return get\_rotate\_labels\_custom(filename, size);

}//..

//rotate

matrix load\_rotate\_labels\_paths(char\*\* paths, int n, float\* labels, int k, tree\* hierarchy, float label\_smooth\_eps, int contrastive)

{

matrix y = make\_matrix(n, 1);

int i;

int j;

int idx;

if (labels) {

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

char\* p = *strrchr*(paths[j], '\\');

idx = *atoi*(p + 1);

y.vals[j] = &labels[idx];

}

}

return y;

}

//rotate

matrix load\_rotate\_labels\_paths(char\*\* paths, int n, float\* labels, int k, tree\* hierarchy, float label\_smooth\_eps, int contrastive)

{

matrix y = make\_matrix(n, 1);

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

fill\_truth\_rotate (paths, n, y.vals[i], labels);

}

return y;

}

void fill\_truth\_rotate (char\*\* paths, int n, float\* truth, float\* labels) {

int j;

int idx;

if (labels) {

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

char\* p = *strrchr*(paths[j], '\\');

idx = *atoi*(p + 1);

truth[j] = labels[idx];

}

}

}

//rotate

data load\_data\_rotate(char\*\* paths, int n, int m, float\* labels, int k, tree\* hierarchy, int use\_flip, int min, int max, int w, int h, float angle,

float aspect, float hue, float saturation, float exposure, int use\_mixup, int use\_blur, int show\_imgs, float label\_smooth\_eps, int dontuse\_opencv, int contrastive)

{

char\*\* paths\_stored = paths;

if (m) paths = get\_random\_paths(paths, n, m);

data d = { 0 };

d.shallow = 0;

d.X = load\_image\_augment\_paths(paths, n, use\_flip, min, max, w, h, angle, aspect, hue, saturation, exposure, dontuse\_opencv, contrastive);

d.y = load\_rotate\_labels\_paths(paths, n, labels, k, hierarchy, label\_smooth\_eps, contrastive);

if (use\_mixup && rand\_int(0, 1)) {

char\*\* paths\_mix = get\_random\_paths(paths\_stored, n, m);

data d2 = { 0 };

d2.shallow = 0;

d2.X = load\_image\_augment\_paths(paths\_mix, n, use\_flip, min, max, w, h, angle, aspect, hue, saturation, exposure, dontuse\_opencv, contrastive);

d2.y = load\_labels\_paths(paths\_mix, n, labels, k, hierarchy, label\_smooth\_eps, contrastive);

free(paths\_mix);

data d3 = { 0 };

d3.shallow = 0;

data d4 = { 0 };

d4.shallow = 0;

if (use\_mixup >= 3) {

char\*\* paths\_mix3 = get\_random\_paths(paths\_stored, n, m);

d3.X = load\_image\_augment\_paths(paths\_mix3, n, use\_flip, min, max, w, h, angle, aspect, hue, saturation, exposure, dontuse\_opencv, contrastive);

d3.y = load\_labels\_paths(paths\_mix3, n, labels, k, hierarchy, label\_smooth\_eps, contrastive);

free(paths\_mix3);

char\*\* paths\_mix4 = get\_random\_paths(paths\_stored, n, m);

d4.X = load\_image\_augment\_paths(paths\_mix4, n, use\_flip, min, max, w, h, angle, aspect, hue, saturation, exposure, dontuse\_opencv, contrastive);

d4.y = load\_labels\_paths(paths\_mix4, n, labels, k, hierarchy, label\_smooth\_eps, contrastive);

free(paths\_mix4);

}

// mix

int i, j;

for (i = 0; i < d2.X.rows; ++i) {

int mixup = use\_mixup;

if (use\_mixup == 4) mixup = rand\_int(2, 3); // alternate CutMix and Mosaic

// MixUp -----------------------------------

if (mixup == 1) {

// mix images

for (j = 0; j < d2.X.cols; ++j) {

d.X.vals[i][j] = (d.X.vals[i][j] + d2.X.vals[i][j]) / 2.0f;

}

// mix labels

for (j = 0; j < d2.y.cols; ++j) {

d.y.vals[i][j] = (d.y.vals[i][j] + d2.y.vals[i][j]) / 2.0f;

}

}

// CutMix -----------------------------------

else if (mixup == 2) {

const float min = 0.3; // 0.3\*0.3 = 9%

const float max = 0.8; // 0.8\*0.8 = 64%

const int cut\_w = rand\_int(w \* min, w \* max);

const int cut\_h = rand\_int(h \* min, h \* max);

const int cut\_x = rand\_int(0, w - cut\_w - 1);

const int cut\_y = rand\_int(0, h - cut\_h - 1);

const int left = cut\_x;

const int right = cut\_x + cut\_w;

const int top = cut\_y;

const int bot = cut\_y + cut\_h;

assert(cut\_x >= 0 && cut\_x <= w);

assert(cut\_y >= 0 && cut\_y <= h);

assert(cut\_w >= 0 && cut\_w <= w);

assert(cut\_h >= 0 && cut\_h <= h);

assert(right >= 0 && right <= w);

assert(bot >= 0 && bot <= h);

assert(top <= bot);

assert(left <= right);

const float alpha = (float)(cut\_w \* cut\_h) / (float)(w \* h);

const float beta = 1 - alpha;

int c, x, y;

for (c = 0; c < 3; ++c) {

for (y = top; y < bot; ++y) {

for (x = left; x < right; ++x) {

int j = x + y \* w + c \* w \* h;

d.X.vals[i][j] = d2.X.vals[i][j];

}

}

}

//printf("\n alpha = %f, beta = %f \n", alpha, beta);

// mix labels

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

d.y.vals[i][j] = d.y.vals[i][j] \* beta + d2.y.vals[i][j] \* alpha;

}

}

// Mosaic -----------------------------------

else if (mixup == 3)

{

const float min\_offset = 0.2; // 20%

const int cut\_x = rand\_int(w \* min\_offset, w \* (1 - min\_offset));

const int cut\_y = rand\_int(h \* min\_offset, h \* (1 - min\_offset));

float s1 = (float)(cut\_x \* cut\_y) / (w \* h);

float s2 = (float)((w - cut\_x) \* cut\_y) / (w \* h);

float s3 = (float)(cut\_x \* (h - cut\_y)) / (w \* h);

float s4 = (float)((w - cut\_x) \* (h - cut\_y)) / (w \* h);

int c, x, y;

for (c = 0; c < 3; ++c) {

for (y = 0; y < h; ++y) {

for (x = 0; x < w; ++x) {

int j = x + y \* w + c \* w \* h;

if (x < cut\_x && y < cut\_y) d.X.vals[i][j] = d.X.vals[i][j];

if (x >= cut\_x && y < cut\_y) d.X.vals[i][j] = d2.X.vals[i][j];

if (x < cut\_x && y >= cut\_y) d.X.vals[i][j] = d3.X.vals[i][j];

if (x >= cut\_x && y >= cut\_y) d.X.vals[i][j] = d4.X.vals[i][j];

}

}

}

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

const float max\_s = 1;// max\_val\_cmp(s1, max\_val\_cmp(s2, max\_val\_cmp(s3, s4)));

d.y.vals[i][j] = d.y.vals[i][j] \* s1 / max\_s + d2.y.vals[i][j] \* s2 / max\_s + d3.y.vals[i][j] \* s3 / max\_s + d4.y.vals[i][j] \* s4 / max\_s;

}

}

}

free\_data(d2);

if (use\_mixup >= 3) {

free\_data(d3);

free\_data(d4);

}

}

#ifdef OPENCV

if (use\_blur) {

int i;

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

if (random\_gen() % 4 == 0) {

image im = make\_empty\_image(w, h, 3);

im.data = d.X.vals[i];

int ksize = use\_blur;

if (use\_blur == 1) ksize = 15;

image blurred = blur\_image(im, ksize);

free\_image(im);

d.X.vals[i] = blurred.data;

//if (i == 0) {

// show\_image(im, "Not blurred");

// show\_image(blurred, "blurred");

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

//}

}

}

}

#endif // OPENCV

if (show\_imgs) {

int i, j;

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

image im = make\_empty\_image(w, h, 3);

im.data = d.X.vals[i];

char buff[1000];

sprintf(buff, "aug\_%d\_%s\_%d", i, basecfg((char\*)paths[i]), random\_gen());

save\_image(im, buff);

char buff\_string[1000];

sprintf(buff\_string, "\n Classes: ");

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

if (d.y.vals[i][j] > 0) {

char buff\_tmp[100];

sprintf(buff\_tmp, " %d (%f), ", j, d.y.vals[i][j]);

strcat(buff\_string, buff\_tmp);

}

}

printf("%s \n", buff\_string);

if (show\_imgs == 1) {

show\_image(im, buff);

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

}

}

printf("\nYou use flag -show\_imgs, so will be saved aug\_...jpg images. Click on window and press ESC button \n");

}

if (m) free(paths);

return d;

}

//rotate

matrix load\_rotate\_labels\_paths(char\*\* paths, int n, float\* labels, int k, tree\* hierarchy, float label\_smooth\_eps, int contrastive)

{

matrix y = make\_matrix(n, 1);

if (labels) {

y.vals[0] = labels;

}

return y;

}

//rotate

matrix load\_rotate\_image\_augment\_paths(char \*\*paths, int n, int use\_flip, int min, int max, int w, int h, float angle, float aspect, float hue, float saturation, float exposure, int dontuse\_opencv, int contrastive)

{

int i;

matrix X;

X.rows = n;

X.vals = (float\*\*)xcalloc(X.rows, sizeof(float\*));

X.cols = 0;

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

int size = w > h ? w : h;

image im;

const int img\_index = (contrastive) ? (i / 2) : i;

if(dontuse\_opencv) im = load\_image\_stb\_resize(paths[img\_index], 0, 0, 3);

else im = load\_image\_gray(paths[img\_index], 0, 0);

image crop = random\_augment\_image(im, angle, aspect, min, max, size);

int flip = use\_flip ? random\_gen() % 2 : 0;

if (flip)

flip\_image(crop);

random\_distort\_image(crop, hue, saturation, exposure);

image sized = resize\_image(crop, w, h);

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

show\_image(sized, "sized");

show\_image(sized, paths[img\_index]);

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

printf("w = %d, h = %d \n", sized.w, sized.h);\*/

free\_image(im);

free\_image(crop);

X.vals[i] = sized.data;

X.cols = sized.h\*sized.w\*sized.c;

}

return X;

}

**Data.h中添加：**

char\*\* get\_rotate\_labels\_custom(char\* filename, int\* size);//rotate

char\*\* get\_rotate\_labels(char\* filename);//rotate

//rotate

data load\_data\_rotate(char\*\* paths, int n, int m, char\*\* labels, int k, tree\* hierarchy, int use\_flip, int min, int max, int w, int h, float angle,

float aspect, float hue, float saturation, float exposure, int use\_mixup, int use\_blur, int show\_imgs, float label\_smooth\_eps, int dontuse\_opencv, int contrastive);

//rotate

matrix load\_image\_augment\_paths(char\*\* paths, int n, int use\_flip, int min, int max, int w, int h, float angle, float aspect, float hue, float saturation, float exposure, int dontuse\_opencv, int contrastive)

**matrix.c中添加**

matrix make\_rotate\_matrix(int rows, int cols)//..

{

int i;

matrix m;

m.rows = rows;

m.cols = cols;

m.vals = (char\*\*)xcalloc(m.rows, sizeof(float\*));

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

m.vals[i] = (char\*)xcalloc(m.cols, sizeof(float));

}

return m;

}

**Image.c中添加**

image load\_image\_gray(char \*filename, int w, int h)//rotate

{

return load\_image(filename, w, h, 1);

}

**Blas.c中添加**

void rotate\_cpu(int n, float\* pred, float\* truth, float\* delta, float\* error)//rotate

{

int i;

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

float diff = 1 - *cos*(pred[i] - truth[i]);

error[i] = diff \* diff;

delta[i] = diff;

}

**Blas.h中添加**

void rotate\_cpu(int n, float\* pred, float\* truth, float\* delta, float\* error)//rotate

**cost\_layer.c中修改**

COST\_TYPE get\_cost\_type(char \*s)//rotate

{

if (*strcmp*(s, "sse")==0) return SSE;

if (*strcmp*(s, "masked")==0) return MASKED;

if (*strcmp*(s, "smooth")==0) return SMOOTH;

if (*strcmp*(s, "rotate") == 0) return ROTATE;

*fprintf*(*stderr*, "Couldn't find cost type %s, going with SSE\n", s);

return SSE;

}