25122871钟浩天 CNN图像识别 详细代码见haotianblog.com

```c
#define IMG_C 3
#define IMG_H 32
#define IMG_W 32
#define IMG_SIZE (IMG_C * IMG_H * IMG_W)
#define NCLASS 10

#define NF 16
#define K 3
#define CONV_H 30
#define CONV_W 30
#define POOL_H 15
#define POOL_W 15
#define FEAT (NF * POOL_H * POOL_W)

#define LR 0.01f
#define MAX_TRAIN 50000
#define MAX_TEST 10000

typedef struct {
    uint8_t label;
    float x[IMG_SIZE];
} Sample;

typedef struct {
    float conv_w[NF][IMG_C][K][K];
    float conv_b[NF];
    float fc_w[NCLASS][FEAT];
    float fc_b[NCLASS];
} Net;

static float frand_uniform(float a, float b) {
    return a + (b - a) * ((float)rand() / (float)RAND_MAX);
}

static void init_net(Net *net) {
    srand(1);
    float conv_scale = sqrtf(2.0f / (IMG_C * K * K));
    for (int f = 0; f < NF; ++f) {
        net->conv_b[f] = 0.0f;
        for (int c = 0; c < IMG_C; ++c)
            for (int r = 0; r < K; ++r)
                for (int s = 0; s < K; ++s)
                    net->conv_w[f][c][r][s] = frand_uniform(-conv_scale, conv_scale);
    }

    float fc_scale = sqrtf(2.0f / FEAT);
    for (int k = 0; k < NCLASS; ++k) {
        net->fc_b[k] = 0.0f;
        for (int i = 0; i < FEAT; ++i)
            net->fc_w[k][i] = frand_uniform(-fc_scale, fc_scale);
    }
}

static int load_batch(const char *filename, Sample *dst, int offset, int max_count) {
    FILE *fp = fopen(filename, "rb");
    if (!fp) {
        fprintf(stderr, "Cannot open %s\n", filename);
        return -1;
    }

    int count = 0;
    while (count < max_count) {
        int label = fgetc(fp);
        if (label == EOF) break;
        dst[offset + count].label = (uint8_t)label;

        uint8_t buf[IMG_SIZE];
        size_t got = fread(buf, 1, IMG_SIZE, fp);
        if (got != IMG_SIZE) break;

        /* CIFAR binary order: label, then 1024 R, 1024 G, 1024 B */
        for (int i = 0; i < IMG_SIZE; ++i) {
            dst[offset + count].x[i] = ((float)buf[i] / 255.0f - 0.5f) / 0.5f;
        }
        count++;
    }
```

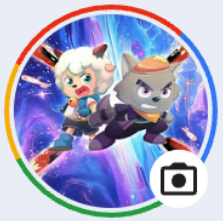
```c
static int load_train(const char *dir, Sample *train, int limit) {
    int total = 0;
    for (int b = 1; b <= 5 && total < limit; ++b) {
        char path[512];
        snprintf(path, sizeof(path), "%s/data_batch_%d.bin", dir, b);
        int need = limit - total;
        if (need > 10000) need = 10000;
        int n = load_batch(path, train, total, need);
        if (n < 0) return -1;
        total += n;
    }
    return total;
}

static int load_test(const char *dir, Sample *test, int limit) {
    char path[512];
    snprintf(path, sizeof(path), "%s/test_batch.bin", dir);
    return load_batch(path, test, 0, limit);
}

static inline float get_pixel(const float *x, int c, int h, int w) {
    return x[c * IMG_H * IMG_W + h * IMG_W + w];
}
```

```c
int p = 0;
for (int f = 0; f < NF; ++f) {
    for (int i = 0; i < POOL_H; ++i) {
        for (int j = 0; j < POOL_W; ++j) {
            int base_i = 2 * i, base_j = 2 * j;
            float best = conv[f][base_i][base_j];
            int best_idx = 0;
            for (int di = 0; di < 2; ++di) {
                for (int dj = 0; dj < 2; ++dj) {
                    int idx = di * 2 + dj;
                    float v = conv[f][base_i + di][base_j + dj];
                    if (v > best) {
                        best = v;
                        best_idx = idx;
                    }
                }
            }
            pool[f][i][j] = best;
            argmax_idx[f][i][j] = best_idx;
            feat[p++] = best;
        }
    }
}

for (int k = 0; k < NCLASS; ++k) {
    float z = net->fc_b[k];
    for (int i = 0; i < FEAT; ++i) z += net->fc_w[k][i] * feat[i];
    logits[k] = z;
}
```

```c
for (int k = 0; k < NCLASS; ++k) {
    float z = net->fc_b[k];
    for (int i = 0; i < FEAT; ++i) z += net->fc_w[k][i] * feat[i];
    logits[k] = z;
}

float maxz = logits[0];
for (int k = 1; k < NCLASS; ++k) if (logits[k] > maxz) maxz = logits[k];
float denom = 0.0f;
for (int k = 0; k < NCLASS; ++k) {
    prob[k] = expf(logits[k] - maxz);
    denom += prob[k];
}
for (int k = 0; k < NCLASS; ++k) prob[k] /= denom;
}

static int predict(const Net *net, const Sample *s) {
    static float conv[NF][CONV_H][CONV_W];
    static float pool[NF][POOL_H][POOL_W];
    static int argmax_idx[NF][POOL_H][POOL_W];
    static float feat[FEAT], logits[NCLASS], prob[NCLASS];
    forward(net, s, conv, pool, argmax_idx, feat, logits, prob);
    int best = 0;
    for (int k = 1; k < NCLASS; ++k) if (prob[k] > prob[best]) best = k;
    return best;
}
```

```c
static float train_one(Net *net, const Sample *s) {
    static float conv[NF][CONV_H][CONV_W];
    static float pool[NF][POOL_H][POOL_W];
    static int argmax_idx[NF][POOL_H][POOL_W];
    static float feat[FEAT], logits[NCLASS], prob[NCLASS];

    forward(net, s, conv, pool, argmax_idx, feat, logits, prob);
    int y = s->label;
    float loss = -logf(prob[y] + 1e-8f);

    float dz[NCLASS];
    for (int k = 0; k < NCLASS; ++k) dz[k] = prob[k] - (k == y ? 1.0f : 0.0f);

    static float dfeat[FEAT];
    memset(dfeat, 0, sizeof(dfeat));

    for (int k = 0; k < NCLASS; ++k) {
        for (int i = 0; i < FEAT; ++i) {
            dfeat[i] += net->fc_w[k][i] * dz[k];
        }
    }

    for (int k = 0; k < NCLASS; ++k) {
        for (int i = 0; i < FEAT; ++i) {
            net->fc_w[k][i] -= LR * dz[k] * feat[i];
        }
        net->fc_b[k] -= LR * dz[k];
    }

    static float dconv[NF][CONV_H][CONV_W];
    memset(dconv, 0, sizeof(dconv));

    for (int f = 0; f < NF; ++f) {
        float db = 0.0f;
        for (int i = 0; i < CONV_H; ++i) {
            for (int j = 0; j < CONV_W; ++j) {
                if (conv[f][i][j] <= 0.0f) dconv[f][i][j] = 0.0f; /* ReLU back
                db += dconv[f][i][j];
            }
        }
        for (int c = 0; c < IMG_C; ++c) {
            for (int r = 0; r < K; ++r) {
                for (int t = 0; t < K; ++t) {
                    float dw = 0.0f;
                    for (int i = 0; i < CONV_H; ++i)
                        for (int j = 0; j < CONV_W; ++j)
                            dw += dconv[f][i][j] * get_pixel(s->x, c, i + r, j + t);
                    net->conv_w[f][c][r][t] -= LR * dw;
                }
            }
        }
        net->conv_b[f] -= LR * db;
    }

    return loss;
}
int main(int argc, char **argv) {
    const char *dir = argc > 1 ? argv[1] : "./cifar-10-batches-bin";
    int epochs = argc > 2 ? atoi(argv[2]) : 1;
    int train_limit = argc > 3 ? atoi(argv[3]) : 5000;
    int test_limit = argc > 4 ? atoi(argv[4]) : 1000;
    if (train_limit > MAX_TRAIN) train_limit = MAX_TRAIN;
    if (test_limit > MAX_TEST) test_limit = MAX_TEST;

    Sample *train = (Sample *)malloc(sizeof(Sample) * train_limit);
    Sample *test = (Sample *)malloc(sizeof(Sample) * test_limit);
    if (!train || !test) {
        fprintf(stderr, "Memory allocation failed.\n");
        return 1;
    }

    int ntrain = load_train(dir, train, train_limit);
    int ntest = load_test(dir, test, test_limit);
    if (ntrain <= 0 || ntest <= 0) {
        fprintf(stderr, "Failed to load CIFAR-10 data. dir=%s\n", dir);
        return 1;
    }
    printf("Loaded train=%d, test=%d\n", ntrain, ntest);
```

zhonghaotian1219@gmail.com



输出结果

```
epoch 10 step 1000/20000 loss=0.8131 train_acc_recent_est=0.927
epoch 10 step 2000/20000 loss=0.8545 train_acc_recent_est=0.934
epoch 10 step 3000/20000 loss=0.8336 train_acc_recent_est=0.938
epoch 10 step 4000/20000 loss=0.8491 train_acc_recent_est=0.939
epoch 10 step 5000/20000 loss=0.8754 train_acc_recent_est=0.939
epoch 10 step 6000/20000 loss=0.8891 train_acc_recent_est=0.938
epoch 10 step 7000/20000 loss=0.9095 train_acc_recent_est=0.937
epoch 10 step 8000/20000 loss=0.9233 train_acc_recent_est=0.937
epoch 10 step 9000/20000 loss=0.9327 train_acc_recent_est=0.936
epoch 10 step 10000/20000 loss=0.9488 train_acc_recent_est=0.934
epoch 10 step 11000/20000 loss=0.9592 train_acc_recent_est=0.934
epoch 10 step 12000/20000 loss=0.9576 train_acc_recent_est=0.933
epoch 10 step 13000/20000 loss=0.9659 train_acc_recent_est=0.933
epoch 10 step 14000/20000 loss=0.9693 train_acc_recent_est=0.932
epoch 10 step 15000/20000 loss=0.9753 train_acc_recent_est=0.932
epoch 10 step 16000/20000 loss=0.9830 train_acc_recent_est=0.931
epoch 10 step 17000/20000 loss=0.9932 train_acc_recent_est=0.929
epoch 10 step 18000/20000 loss=1.0006 train_acc_recent_est=0.928
epoch 10 step 19000/20000 loss=1.0083 train_acc_recent_est=0.927
epoch 10 step 20000/20000 loss=1.0141 train_acc_recent_est=0.926
epoch 10 done: avg_loss=1.0141 train_acc=0.926 test_acc=0.470
Model saved to model_weights.bin
```

类别编号	类别	样本数	正确数	正确率
0	airplane	488	221	45.29%
1	automobile	505	309	61.19%
2	bird	512	183	35.74%
3	cat	497	138	27.77%
4	deer	507	254	50.10%
5	dog	488	185	37.91%
6	frog	491	251	51.12%
7	horse	495	258	52.12%
8	ship	504	280	55.56%
9	truck	513	272	53.02%