报告封面）



“数据结构”

课程设计报告

**设计题目**  马踏棋盘 CNN开放题

**姓 名**  付炎平

**学 号**  2019217819

**专 业**  物联网工程

**班 级**

**完成日期**

**马踏棋盘**

1. **需求和规格说明**

将马随机放在国际象棋的 8\* 8 棋盘 Bord[8Ⅱ8]的某个方格中，

马按走棋规则进行移动。要求每个方格上只进入一次，走遍棋盘上全部 64 个方

格。

任务要求：编制非递归程序，求出马的行走路线 ，并按求出的行走路线，

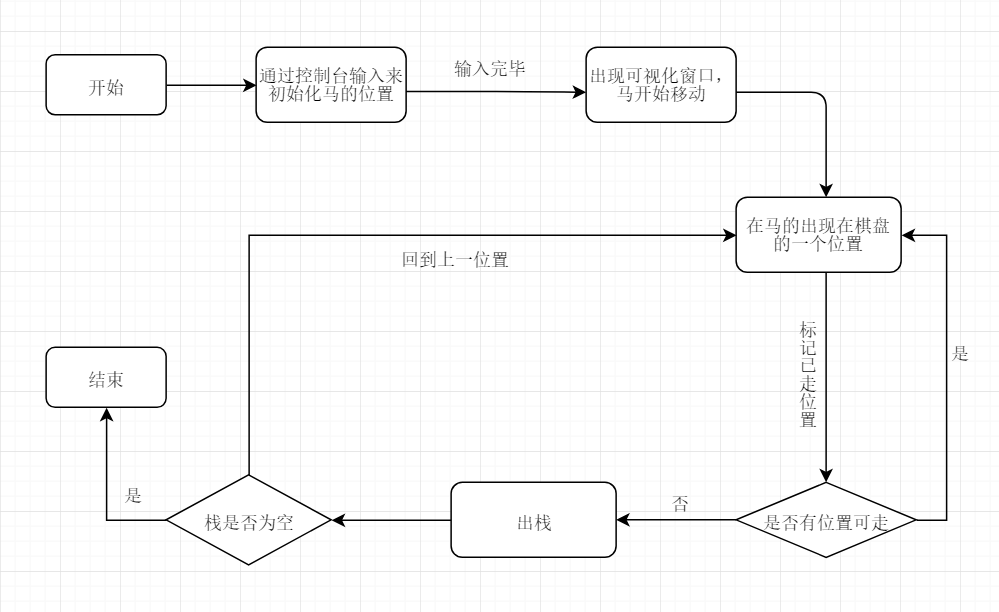
将数字 1，2，…，64 依次填入一个 8\*8 的方阵，输出之。

**（二） 设计**

根据上述要求，首先可以用一个8\*8的二维数组来储存棋盘，并用一个x,y的坐标来储存马的位置，由于马要按国际象棋的标准进行移动，所以采用一个step数组来表示马的不能方向的移动。

由于题目要求不得用递归程序，故可以采用栈来进行编程，利用入栈把马的位置进行储存起来，利用出栈来表示回溯，最终当马把整齐棋盘都踏完的时候，也就是栈为空的时候。

由于题目要求马的行走路线，故我采用创建可视化窗口来对马的行走进行一个动态展示，并标记马走的每一步。



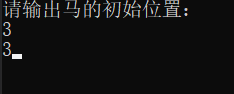
**（三） 用户手册**

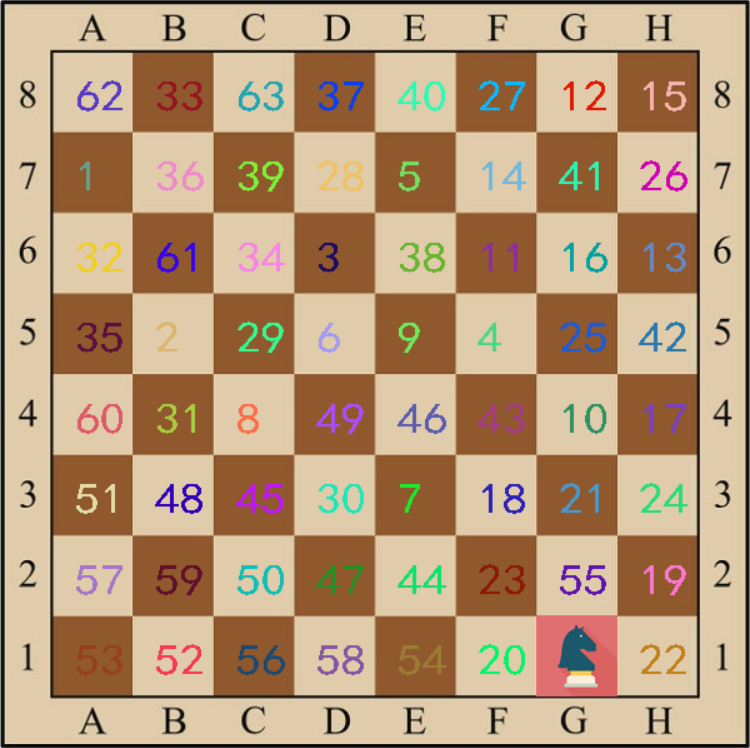
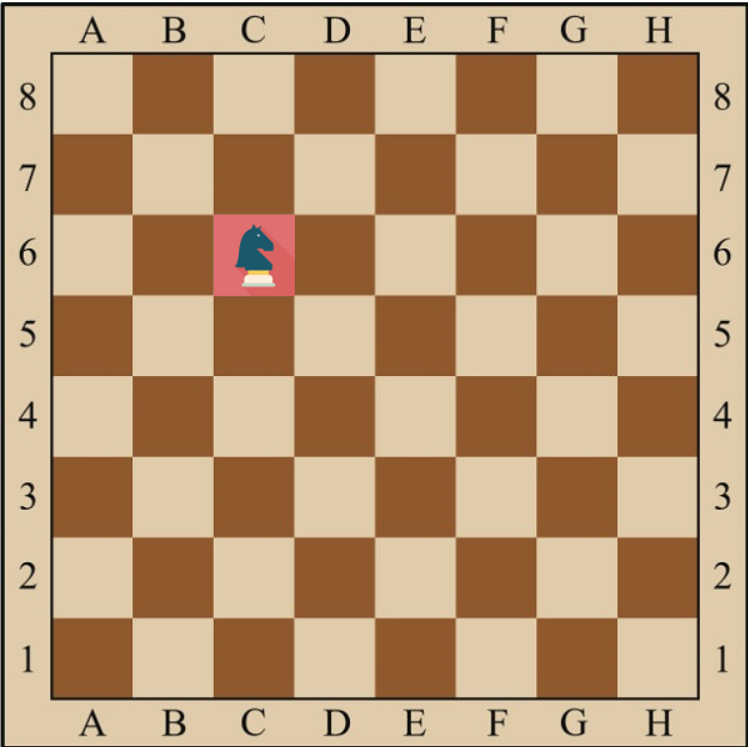
程序运行时，首先提示用户输出一个坐标来作为马的初始位置，然后只需观察可视化棋盘上马的移动和马走过位置的标记。

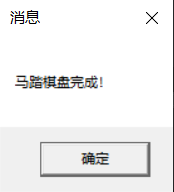
**（四） 调试及测试**

由于需要用栈来进行储存，故需要掌握元素入栈和出栈的时机，并把马的坐标用一个结构体来表示。又因为需要创建可视化窗口，所以需要不断调整马的坐标来和棋盘上相匹配，同也需要调整数字的位置来和棋盘上网格相匹配。

1. **运行实例：**







**（六）进一步改进**

由于是需要创建可视化窗口，我认为我这个和用户进行交互的功能也不是很齐全，还有程序对于路线设计的可视化仍然需要进一步改进。

**（七）心得体会**

我认为这次课设让我对可视化操作有了进一步的了解，并且对于用栈来消除递归也具有一定的了解，并且对于深度优先搜索算法也有了深刻的认识，并且提高了我写代码的能力。

**（八）对课程设计的建议**

我认为课程设计给我们学生提供了和一个展示自己编程能力的机会，并且能够激发我们编程的兴趣，让我们明白编程其实还是有很大的作用的。

**（九）附录⎯⎯源程序**

#include <stack>

#include <iostream>

#include<time.h>

#include<graphics.h>

using namespace std;

struct node

{

int x;

int y;

};

IMAGE img1, img2;

int map[8][8] = { 0 };

int step[8][2] = { {-2,-1} ,{-2,1},{2,-1},{2,1},{-1,-2},{-1,2},{1,-2},{1,2} };

int visit[8][8] = { 0 };

bool bound(int x, int y) {

if (x < 8 && x>=0 && y < 8 && y>=0 ) {

return true;

}

return false;

}

void draw\_number() {

for (int i = 0; i < 8; i++) {

for (int j = 0; j < 8; j++) {

if (map[i][j] != 0) {

//设置字体的颜色

//当它行数加上列数为奇数的时候

if ((i + j) % 2 == 1) {

setbkcolor(RGB(143,89,45));

}

//当它行数加上列数为偶数的时候

else {

setbkcolor(RGB(224,204,171));

}

//把字体进行可视化

char ch[10] = " ";

sprintf\_s(ch, "%d", map[i][j]);

settextstyle(35, 20, "楷体");

settextcolor(RGB(rand() % 256, rand() % 256, rand() % 256, ));

outtextxy( 60+ 64.5 \* j, 60 + 64.5 \* i, ch);

}

}

}

}

int find\_path(int x,int y) {

//设置随机数种子

srand((unsigned)time(NULL));

//用栈来储存结构体

stack<node>s;

node p;

p.x = x;

p.y = y;//初始化马的位置（x=1，y=0）

s.push(p);

visit[p.x][p.y] = 1;

int num = 1; //进行初始化

map[p.x][p.y] = num;

putimage(0, 0, &img1);

draw\_number();

putimage(42 + 64.5 \* p.y, 42 + 64.5 \* p.x, &img2);

Sleep(1000);

while (!s.empty()) {

int t = 0; //逻辑判断，无路可走情况

for (int i = 0; i < 8; i++) {

p = s.top();

node h = p;

h.x = p.x + step[i][0];

h.y = p.y + step[i][1];

if (visit[h.x][h.y] == 0 && bound(h.x, h.y)) {//判断位置是否有效

//可视化棋盘

putimage(0, 0, &img1);

draw\_number();

putimage(42 + 64.5 \* h.y, 42 + 64.5 \* h.x, &img2);

s.push(h);

//当马每踏过一个网格就把数字num+1

num++;

map[h.x][h.y] = num;

visit[h.x][h.y] = 1;

t = 1;

//间隔一秒便于观察

Sleep(1000);

}

}

if (t == 0) {

s.pop();//走到尽头回溯

}

}

return 0;

}

int main()

{

//创建可视化窗户

loadimage(&img1, "1.jpg");

loadimage(&img2, "2.png");

//开始马踏棋盘

int x;

int y;

cout << "请输出马的初始位置："<<endl;

cin >> x;

cin >> y;

initgraph(600, 600);

find\_path(x-1,y-1);

//结束时候的标志

MessageBox(NULL, "马踏棋盘完成！", "消息", MB\_OK);

closegraph();

}

CNN开放题

**（一） 需求和规格说明**

由于这道题是CNN开放题经过与老师商定后，决定用C++做一个分类器，采用全连接神经网络对几个不同矩阵进行分类。

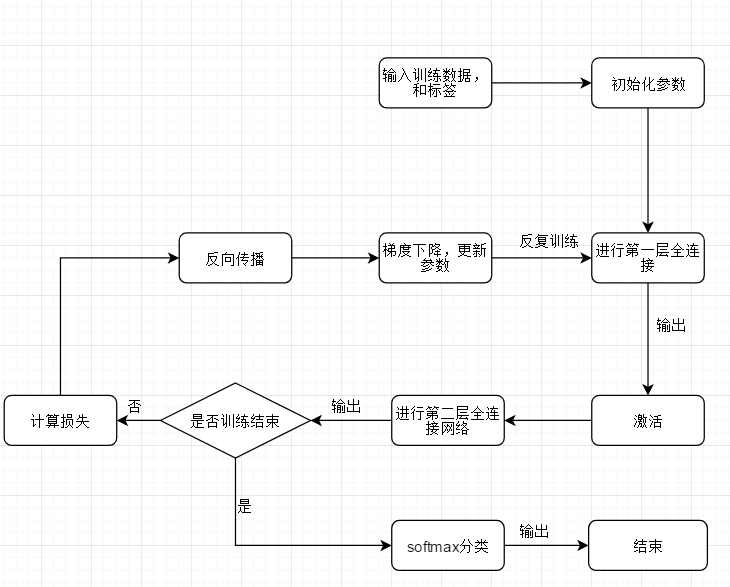
采用C++语言，利用STL中的vector来表示矩阵，通过自己编写类来实现神经网络的操作。

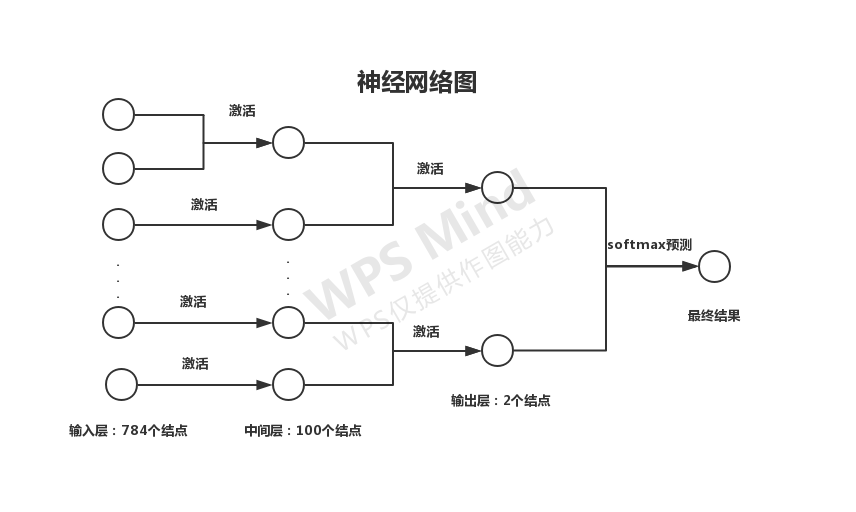
**（二） 设计**

根据上述要求，设计一个类，初始化需要学习的参数，例如w1,w2,d1,d2,初始化参数的维度是由输入数据和输出数据的大小来决定的，由于神经网络里面的计算是通过矩阵计算的，故需要自己编写成员函数来矩阵操作，除了矩阵相乘，还需要自己实现的是矩阵相加，相减，矩阵与数字相乘，相除。

实现完这一系列基本操作后，需要自己编写前向传播函数，通过opencv把图片转换成矩阵，进入转换成张量作为输入数据，再把输入数据通过与参数进行运算实现2个隐层的神经网络，在每次输出预测后与标签进行对比，计算出损失。

计算完损失之后，还要进行反向传播和梯度下降来更新参数，反向传播需要自己手动求导，通过不断训练学习一个合适的参数使得能够准确预测。





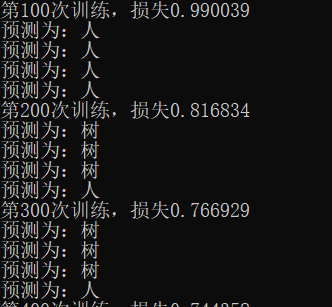
**（三） 用户手册**

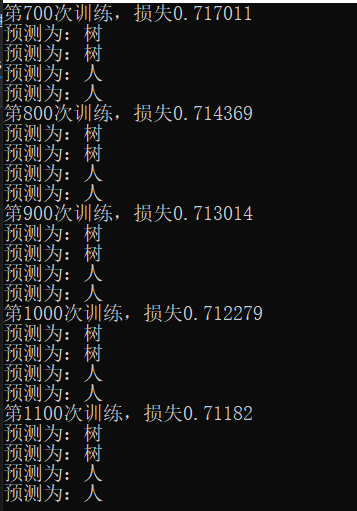
需要提供训练数据和标签来进行训练，并间隔查看损失，发现损失在不断减小，当损失越小的时候，训练的参数也就越准确，最后把训练的参数和测试数据进行预测，就会发现预测的很准确。

**（四） 调试及测试**

由于需要进行矩阵相乘，所以前向传播、反向传播和梯度更新时候需要自己不断调试，直到矩阵的维度相匹配时，不然就会出现数组越界的错误。在训练的时候，需要自己不断调试设置一个比较好的学习率来提高训练的效率。

**（五） 运行实例：**





1. **进一步改进**

由于这需要自己手动写函数，所以处理图片多分类的时会执行的很慢，所以只能通过进行比较少的分类来完成，其次没有完成卷积神经网络的实现，可能因为卷积神经网络非常复杂，以及反向传播自己手写函数十分困难，所以这是需要改进的地方。

1. **心得体会**

我认为这次课设难度比较大，充分考验了我对神经网络理论的知识，并对C++掌握的熟练度，我在这次课设颇有收获，对于opencv、C++和一些STL中的一些工具都有了更深的理解和认识。

1. **对课程设计的建议**

我认为这次课设能够充分的锻炼学生的编程能力，对于课程设计这一安排我认为非常合适，能够让学生学会运用所学的编程知识。

1. **附录⎯⎯源程序**

#include <vector>

#include <iostream>

#include <stdlib.h>

#include <time.h>

#include <math.h>

#include <string.h>

#include <opencv.hpp>

using namespace cv;

using namespace std;

class Linear { //定义全连接神经网络类

public:

vector<vector<double> > w1, w2, b1, b2, x, y, y\_pred, A1, A2, h\_relu;

double loss;

Linear(int inchanels,int outchanels,int batch\_size ) { //初始化参数

srand((unsigned)time(NULL));

int mid\_layer = 100;

vector<vector<double> > w1(mid\_layer, vector<double>(inchanels));

for (int i = 0; i < w1.size(); i++) {

for (int j = 0; j < w1[0].size(); j++) {

w1[i][j] = double(rand() % 1000) / 1000/10;

}

}

this->w1 = w1;

vector<vector<double> >b1(mid\_layer, vector<double>(batch\_size));

this->b1 = b1;

vector<vector<double> >w2(outchanels, vector<double>(mid\_layer));

for (int i = 0; i < w2.size(); i++) {

for (int j = 0; j < w2[0].size(); j++) {

w2[i][j] = double(rand() % 1000) / 1000\*0.01;

}

}

this->w2 = w2;

vector<vector<double> >b2(outchanels, vector<double>(batch\_size));

this->b2 = b2;

}

vector<vector<double> > dot(vector<vector<double> > v1, vector<vector<double> > v2) {

vector<vector<double> > v3(v1.size(), vector<double>(v2[0].size()));

double s = 0.00000; //实现矩阵相乘

for (int h = 0; h < v2[0].size(); h++) {

for (int k = 0; k < v1.size(); k++) {

for (int j = 0; j < v2.size(); j++) {

s += v1[k][j] \* v2[j][h];

}

v3[k][h] = s;

s = 0.00000;

}

}

return v3;

}

vector<vector<double> >Relu(vector<vector<double> > v) { //实现Relu激活函数

vector<vector<double> > v3(v.size(), vector<double>(v[0].size()));

for (int i = 0; i < v.size(); i++) {

for (int j = 0; j < v[0].size(); j++) {

if (v[i][j] < 0) {

v3[i][j] = 0;

}

else {

v3[i][j] = v[i][j];

}

}

}

return v3;

}

vector<vector<double> > add(vector<vector<double> > v1, vector<vector<double> > v2) {

vector<vector<double> > v3(v1.size(), vector<double>(v1[0].size()));

for (int i = 0; i < v1.size(); i++) { //实现矩阵相加

for (int j = 0; j < v1[0].size(); j++) {

v3[i][j] = v1[i][j] + v2[i][j];

}

}

return v3;

}

vector<vector<double> > subtract(vector<vector<double> > v1, vector<vector<double> > v2) {

vector<vector<double> > v3(v1.size(), vector<double>(v1[0].size()));

for (int i = 0; i < v1.size(); i++) { //实现矩阵相减

for (int j = 0; j < v1[0].size(); j++) {

v3[i][j] = v1[i][j] - v2[i][j];

}

}

return v3;

}

vector<vector<double> > softmax(vector<vector<double> > x1) { //实现softmax分类预测

A1 = add(dot(w1, x1), b1);

h\_relu = Relu(A1);

A2 = add(dot(w2, h\_relu), b2);

vector<vector<double> > y\_pre = Exp(A2);

double s = sum(y\_pre);

y\_pre = division(y\_pre, s);

vector<vector<double> > v(1, vector<double>(y\_pre[0].size()));

double max, max\_idx;

for (int j = 0; j < y\_pre[0].size(); j++) {

max = y\_pre[0][j];

max\_idx = 0;

for (int i = 1; i < y\_pre.size(); i++) {

if (y\_pre[i][j] > max) {

max = y\_pre[i][j];

max\_idx = i;

}

}

v[0][j] = max\_idx;

}

return v;

}

vector<vector<double> >multiply(vector<vector<double> > v1, vector<vector<double> > v2) {

vector<vector<double> > v3(v1.size(), vector<double>(v1[0].size()));

for (int i = 0; i < v1.size(); i++) { //实现矩阵的各元素分别相乘

for (int j = 0; j < v1[0].size(); j++) {

v3[i][j] = v1[i][j] \* v2[i][j];

}

}

return v3;

}

vector<vector<double> >mult\_num(vector<vector<double> > v, double num) {

vector<vector<double> > v3(v.size(), vector<double>(v[0].size()));

for (int i = 0; i < v.size(); i++) { //实现矩阵与数相乘

for (int j = 0; j < v[0].size(); j++) {

v3[i][j] = v[i][j] \* num;

}

}

return v3;

}

vector<vector<double> >Exp(vector<vector<double> > v) { //实现矩阵的exp函数

vector<vector<double> > v3(v.size(), vector<double>(v[0].size()));

for (int i = 0; i < v.size(); i++) {

for (int j = 0; j < v[0].size(); j++) {

v3[i][j] = exp(v[i][j]);

}

}

return v3;

}

vector<vector<double> >Log(vector<vector<double> > v) { //实现矩阵的log函数

vector<vector<double> > v3(v.size(), vector<double>(v[0].size()));

for (int i = 0; i < v.size(); i++) {

for (int j = 0; j < v[0].size(); j++) {

v3[i][j] = log(v[i][j]);

}

}

return v3;

}

vector<vector<double> >division(vector<vector<double> > v, double num) {

vector<vector<double> > v3(v.size(), vector<double>(v[0].size()));

for (int i = 0; i < v.size(); i++) { //实现矩阵与数相除

for (int j = 0; j < v[0].size(); j++) {

v3[i][j] = v[i][j] / num;

}

}

return v3;

}

double sum(vector<vector<double> > v) {

double s = 0.00000000; //实现矩阵中各元素求和

for (int i = 0; i < v.size(); i++) {

for (int j = 0; j < v[0].size(); j++) {

s += v[i][j];

}

}

return s;

}

vector<vector<double> > T(vector<vector<double> > v) { //实现矩阵的转置

vector<vector<double> > v3(v[0].size(), vector<double>(v.size()));

for (int i = 0; i < v.size(); i++) {

for (int j = 0; j < v[0].size(); j++) {

v3[j][i] = v[i][j];

}

}

return v3;

}

void forward(vector<vector<double> > x, vector<vector<double> > y) {

this->x = x; //实现前向传播函数

this->y = y;

A1 = add(dot(w1, x), b1);

h\_relu = Relu(A1);

A2 = add(dot(w2, h\_relu), b2);

y\_pred = Exp(A2);

double s = sum(y\_pred);

y\_pred = division(y\_pred, s);

loss = -sum(multiply(y, Log(y\_pred)));

}

void optimization(double alpha) { //实现反向传播和优化函数

vector<vector<double> > dA2 = subtract(y\_pred, y);

vector<vector<double> > dw2 = dot(dA2, T(A1));

vector<vector<double> > db2 = dA2; //反向传播

vector<vector<double> > dh\_relu = dot(T(w2), dA2);

vector<vector<double> > dA1 = dh\_relu;

for (int i = 0; i < A1.size(); i++) {

for (int j = 0; j < A1[0].size(); j++) {

if (A1[i][j] < 0) {

dA1[i][j] = 0;

}

}

}

vector<vector<double> > dw1 = dot(dA1, T(x));

vector<vector<double> > db1 = dA1;

w1 = subtract(w1, mult\_num(dw1, alpha)); //优化

b1 = subtract(b1, mult\_num(db1, alpha));

w2 = subtract(w2, mult\_num(dw2, alpha));

b2 = subtract(b2, mult\_num(db2, alpha));

}

};

vector<vector<double> > decode(string path) //把图片转换为矩阵

{

Mat img = imread(path,0);

Mat dst;

int w = img.cols \* img.channels(); //可能为3通道，宽度要乘图片的通道数

int h = img.rows;

vector<vector<double> > array(h, vector<double>(w)); //初始化二维vector

for (int i = 0; i < h; i++)

{

uchar\* inData = img.ptr<uchar>(i); //ptr为指向图片的行指针，参数i为行数

for (int j = 0; j < w; j++)

{

array[i][j] = ((double)inData[j]);

}

}

return array;

}

vector<vector<double> > Tensor(vector<vector<double> > v) { //把矩阵转换为张量

int size = v.size() \* v[0].size();

vector<vector<double> > v3(size, vector<double>(1));

int k = 0;

for (int i = 0; i < v.size(); i++) {

for (int j = 0; j < v[0].size(); j++) {

v3[k][0] = v[i][j];

}

}

return v3;

}

int main() {

string path[2][5] = { //训练图片路径

{"train/1.jpg","train/2.jpg","train/3.jpg","train/4.jpg","train/5.jpg"},

{"train/11.jpg","train/12.jfif","train/13.jpg","train/14.jpg","train/15.jpg"},

};

string predict\_path[4] = { //测试图片路径

"predict/1.jpg","predict/2.jpg",

"predict/11.jpg","predict/12.jpg",

};

vector<vector<double> > v = Tensor(decode(path[0][0]));

int inchanels = v.size();

int outchanels = 2;

int batch\_size = v[0].size();

Linear Lin(inchanels, outchanels, batch\_size);

for (int i = 0; i < 2000; i++) {

vector<vector<double> > x = Tensor(decode(path[i % 2][i % 5]));

vector<vector<double> > y(outchanels, vector<double>(1));

y[i % 2][0] = 1;

Lin.forward(x, y); //前向传播

Lin.optimization(0.001); //反向传播，梯度下降

if (i % 100 == 99) {

cout << "第" << i + 1 << "次训练，" << "损失" << Lin.loss << endl;

for (int k = 0; k < 4; k++) {

vector < vector<double> > x1 = Tensor(decode(predict\_path[k]));

vector<vector<double> >pred = Lin.softmax(x1);//每100次的时候进行预测

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

for (int j = 0; j < pred[0].size(); j++) {

if (pred[i][j] == 0) {

cout << "预测为：" << "树" << endl;

}

if (pred[i][j] == 1) {

cout << "预测为：" << "人" << endl;

}

}

}

}

}

}

}