# OpenCV中神经网络的使用

由于OpenCV中神经网络与其它机器学习分类器使用略有不同，特别对神经网络使用进行说明，以供以后参考，少走弯路。

## 使用说明

1. 神经网络的创建

神经网络创建需要先定义神经网络的层次结构，用一个整型一维矩阵表示，如：

*Mat* layer\_sizes(1, 3, *CV\_32SC1*);

layer\_sizes.*at*<int>(0) = 2; //输入的特征个数，即每个样本特征的个数

layer\_sizes.*at*<int>(1) = 5; //隐藏层神经元的个数

layer\_sizes.*at*<int>(2) = 3; //输出层的输出个数，即类别数量

既可以先定义一个神经网络对象，再使用create方法创建，也可以在定义时传入结构参数，如：

*CvANN\_MLP* ann;

ann.*create*(layer\_sizes);

或者：

*CvANN\_MLP* ann(layer\_sizes);

1. 神经网络的训练

与其它分类器一样，在训练神经网络时，传入特征样本、样本标签，另外还要传入样本的权重。需要注意的是传入参数的类型及格式。

样本特征矩阵：样本数量×特征数量 CV\_32FC1类型

样本标签矩阵：样本数量×分类数量 CV\_32FC1类型，这与其它分类器不同，例如，有三类样本，数量分别为2、2、2，则样本标签矩阵为：

1, 0, 0;

1, 0, 0;

0, 1, 0;

0, 1, 0;

0, 0, 1;

0, 0, 1;

样本权重矩阵：1×样本数量，CV\_32FC1类型，一般都设为1。

1. 神经网络的预测分类

使用predict函数，predict函数原型为：

float CvANN\_MLP::**predict**(const Mat& **inputs**, Mat& **outputs**) const

预测结果保存在outputs矩阵中，矩阵大小为 测试样本数量×分类数量，对于只有一个测试样本，矩阵大小为1×分类数量，如果在创建神经网络时使用默认的触发函数cvANN\_MLP::SIGMOID\_SYM，且fparam1和fparam2为0，则输出矩阵的值在[-1.7159,1.7159]之间，而不是[0,1]，取最大值的位置作为分类结果，例如：

[1.0263157, 0.072333388, -0.02631394]

则取0为预测结果，可以使用下面的代码实现：

ann.*predict*(desc, outputs);

*cv*::*Point* maxLoc;

*cv*::*minMaxLoc*(outputs, 0, 0, 0, &maxLoc);

res = maxLoc.*x*;

## 实例

#include "stdafx.h"

#include <opencv2/opencv.hpp>

#include <stdio.h>

using namespace *std*;

using namespace *cv*;

const *Scalar* WHITE\_COLOR = *Scalar*(255, 255, 255);

const *string* winName = "points";

const int testStep = 5;

*Mat* img, imgDst;

*RNG* rng;

*vector*<*Point*> trainedPoints;

*vector*<int> trainedPointsMarkers;

*vector*<*Scalar*> classColors;

static void on\_mouse(int event, int x, int y, int /\*flags\*/, void\*)

{

if (img.*empty*())

return;

int updateFlag = 0;

if (event == *CV\_EVENT\_LBUTTONUP*)

{

if (classColors.*empty*())

return;

trainedPoints.*push\_back*(*Point*(x, y));

trainedPointsMarkers.*push\_back*((int)(classColors.*size*() - 1));

updateFlag = true;

}

else if (event == *CV\_EVENT\_RBUTTONUP*)

{

classColors.*push\_back*(*Scalar*((*uchar*)rng(256), (*uchar*)rng(256), (*uchar*)rng(256)));

updateFlag = true;

}

//draw

if (updateFlag)

{

img = *Scalar*::*all*(0);

// put the text

*stringstream* text;

text << "current class " << classColors.*size*() - 1;

*putText*(img, text.*str*(), *Point*(10, 25), *FONT\_HERSHEY\_SIMPLEX*, 0.8f, WHITE\_COLOR, 2);

text.*str*("");

text << "total classes " << classColors.*size*();

*putText*(img, text.*str*(), *Point*(10, 50), *FONT\_HERSHEY\_SIMPLEX*, 0.8f, WHITE\_COLOR, 2);

text.*str*("");

text << "total points " << trainedPoints.*size*();

*putText*(img, text.*str*(), *Point*(10, 75), *FONT\_HERSHEY\_SIMPLEX*, 0.8f, WHITE\_COLOR, 2);

// draw points

for (*size\_t* i = 0; i < trainedPoints.*size*(); i++)

*circle*(img, trainedPoints[i], 5, classColors[trainedPointsMarkers[i]], -1);

*imshow*(winName, img);

}

}

static void prepare\_train\_data(*Mat*& samples, *Mat*& classes)

{

*Mat*(trainedPoints).*copyTo*(samples);

*Mat*(trainedPointsMarkers).*copyTo*(classes);

// reshape trainData and change its type

samples = samples.*reshape*(1, samples.*rows*);

samples.*convertTo*(samples, *CV\_32FC1*);

}

void find\_decision\_boundary\_ANN(const *Mat*& layer\_sizes)

{

img.*copyTo*(imgDst);

*Mat* trainSamples, trainClasses;

prepare\_train\_data(trainSamples, trainClasses);

// prerare trainClasses

trainClasses.*create*(trainedPoints.*size*(), classColors.*size*(), *CV\_32FC1*);

for (int i = 0; i < trainClasses.*rows*; i++)

{

for (int k = 0; k < trainClasses.*cols*; k++)

{

if (k == trainedPointsMarkers[i])

trainClasses.*at*<float>(i, k) = 1;

else

trainClasses.*at*<float>(i, k) = 0;

}

}

*Mat* weights(1, trainedPoints.*size*(), *CV\_32FC1*, *Scalar*::*all*(1));

// learn classifier

*CvANN\_MLP* ann(layer\_sizes, *CvANN\_MLP*::*SIGMOID\_SYM*, 1, 1);

ann.*train*(trainSamples, trainClasses, weights);

*Mat* testSample(1, 2, *CV\_32FC1*);

for (int y = 0; y < img.*rows*; y += testStep)

{

for (int x = 0; x < img.*cols*; x += testStep)

{

testSample.*at*<float>(0) = (float)x;

testSample.*at*<float>(1) = (float)y;

*Mat* outputs;

ann.*predict*(testSample, outputs);

*cout* << outputs << *endl*;

*Point* maxLoc;

*minMaxLoc*(outputs, 0, 0, 0, &maxLoc);

*circle*(imgDst, *Point*(x, y), 2, classColors[maxLoc.x], 1);

}

}

}

int *main*()

{

*cout* << "Use:" << *endl*

<< " right mouse button - to add new class;" << *endl*

<< " left mouse button - to add new point;" << *endl*

<< " key 'r' - to run the ML model;" << *endl*

<< " key 'i' - to init (clear) the data." << *endl* << *endl*;

*cv*::*namedWindow*(winName, 1);

img.*create*(480, 640, *CV\_8UC3*);

imgDst.*create*(480, 640, *CV\_8UC3*);

*imshow*(winName, img);

*cvSetMouseCallback*(winName.*c\_str*(), on\_mouse);

for (;;)

{

*uchar* key = (*uchar*)*waitKey*();

if (key == 27) break;

if (key == 'i') // init

{

img = *Scalar*::*all*(0);

classColors.*clear*();

trainedPoints.*clear*();

trainedPointsMarkers.*clear*();

*imshow*(winName, img);

}

if (key == 'r') // run

{

*Mat* layer\_sizes1(1, 3, *CV\_32SC1*);

layer\_sizes1.*at*<int>(0) = 2;

layer\_sizes1.*at*<int>(1) = 5;

layer\_sizes1.*at*<int>(2) = classColors.*size*();

find\_decision\_boundary\_ANN(layer\_sizes1);

*namedWindow*("ANN", *WINDOW\_AUTOSIZE*);

*imshow*("ANN", imgDst);

}

}

return 1;

}