# OpenCV3.3中使用dnn

相比于之前的OpenCV版本，3.3将使用dnn不再需要contrib包，在使用上也进行了一些简化，并增加了一些函数功能。下面封装了CaffeClassifer类，简化使用。

## CaffeClassifer类

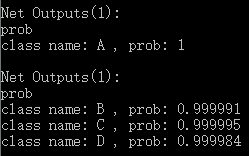
|  |
| --- |
| #include "stdafx.h"  #include <opencv2/dnn.hpp>  #include <opencv2\opencv.hpp>  #include <fstream>  #include <iostream>  #include <cstdlib>  using namespace std;  using namespace cv;  using namespace cv::dnn;  class CaffeClassifier  {  public:  CaffeClassifier(std::string model\_name, std::string net\_name, std::string class\_name);  void setSize(Size \_size);  std::pair<string, double> predict(cv::Mat image);  std::vector<std::pair<string, double>> predict(std::vector<cv::Mat> images);  private:  std::pair<int, double> getMaxClass(cv::Mat prob);  std::vector<std::string> readClassNames(string class\_name);  private:  std::vector<std::string> className;  cv::dnn::Net net;  cv::Size input\_size;  bool bNormal;  };  CaffeClassifier::CaffeClassifier(string model\_name, string net\_name, string class\_name)  {  className = readClassNames(class\_name);  net = dnn::readNetFromCaffe(net\_name, model\_name);  if (net.empty())  {  cout << "load model failed!";  exit(-1);  }  bNormal = false;  }  void CaffeClassifier::setSize(Size \_size)  {  input\_size = \_size;  }  std::pair<int, double> CaffeClassifier::getMaxClass(cv::Mat prob)  {  Point classNumber;  double classProb = 0.0;  minMaxLoc(prob, NULL, &classProb, NULL, &classNumber);  return std::pair<int, double>(classNumber.x, classProb);  }  std::pair<string, double> CaffeClassifier::predict(Mat img)  {  resize(img, img, input\_size);  img.convertTo(img, CV\_32FC3);  img /= 255.0;  net.setInput(blobFromImage(img), "data");  Mat out = net.forward("prob");  std::pair<int, double> res = getMaxClass(out);  return pair<string, double>(className[res.first], res.second);  }  vector<string> CaffeClassifier::readClassNames(string class\_name)  {  std::vector<string> synset\_words;  std::ifstream fp(class\_name);  if (!fp.is\_open())  {  std::cerr << "File with classes labels not found: " << class\_name << std::endl;  exit(-1);  }  std::string name;  while (!fp.eof())  {  std::getline(fp, name);  if (name.length())  synset\_words.push\_back(name.substr(name.find(' ') + 1));  }  fp.close();  return synset\_words;  }  std::vector<std::pair<string, double>> CaffeClassifier::predict(std::vector<cv::Mat> images)  {  vector<pair<string, double>> results;  for (int i = 0;i < images.size();i++)  {  resize(images[i], images[i], input\_size);  images[i].convertTo(images[i], CV\_32FC3);  images[i] /= 255.0;  }  net.setInput(blobFromImages(images), "data");  Mat out = net.forward();  for (int i = 0;i < out.rows;i++)  {  std::pair<int, double> res = getMaxClass(out.row(i));  results.push\_back(pair<string, double>(className[res.first], res.second));  }  return results;  } |

## Demo

### 2.1 主函数

|  |
| --- |
| #define PRIDECT\_RESULT std::pair<std::string, double>  void main()  {  CaffeClassifier classifer("plate\_iter\_5000.caffemodel", "plate\_deloy.prototxt", "dict.txt");  classifer.setSize(cv::Size(28, 28));  //测试一张图像  Mat image = cv::imread("A.bmp", 1);  PRIDECT\_RESULT res = classifer.predict(image);  cout << "class name: " << res.first << " , prob: " << res.second << endl;  imshow("image", image);  waitKey(0);  //测试几张图像  Mat im1 = imread("B.bmp", 1);  Mat im2 = imread("C.bmp", 1);  Mat im3 = imread("D.bmp", 1);  vector<Mat> images;  images.push\_back(im1);  images.push\_back(im2);  images.push\_back(im3);  vector<PRIDECT\_RESULT> ress = classifer.predict(images);  for (int i = 0;i < images.size();i++)  {  cout << "class name: " << ress[i].first << " , prob: " << ress[i].second << endl;  imshow("image", images[i]);  waitKey(0);  }  } |

A B C D



### 2.2 plate\_deloy.prototxt

|  |
| --- |
| name: "PlateNet"  input: "data"  input\_shape {  dim: 1 # batchsize  dim: 3 # number of colour channels - rgb  dim: 28 # width  dim: 28 # height  }  layer {  name: "conv1"  type: "Convolution"  bottom: "data"  top: "conv1"  param {  lr\_mult: 1  }  param {  lr\_mult: 2  }  convolution\_param {  num\_output: 20  kernel\_size: 5  stride: 1  weight\_filler {  type: "xavier"  }  bias\_filler {  type: "constant"  }  }  }  layer {  name: "pool1"  type: "Pooling"  bottom: "conv1"  top: "pool1"  pooling\_param {  pool: MAX  kernel\_size: 2  stride: 2  }  }  layer {  name: "ip1"  type: "InnerProduct"  bottom: "pool1"  top: "ip1"  param {  lr\_mult: 1  }  param {  lr\_mult: 2  }  inner\_product\_param {  num\_output: 500  weight\_filler {  type: "xavier"  }  bias\_filler {  type: "constant"  }  }  }  layer {  name: "relu1"  type: "ReLU"  bottom: "ip1"  top: "ip1"  }  layer {  name: "ip2"  type: "InnerProduct"  bottom: "ip1"  top: "ip2"  param {  lr\_mult: 1  }  param {  lr\_mult: 2  }  inner\_product\_param {  num\_output: 34  weight\_filler {  type: "xavier"  }  bias\_filler {  type: "constant"  }  }  }  layer {  name: "prob"  type: "Softmax"  bottom: "ip2"  top: "prob"  } |

### 2.3 dict.txt

|  |
| --- |
| 0 0  1 1  2 2  3 3  4 4  5 5  6 6  7 7  8 8  9 9  10 A  11 B  12 C  13 D  14 E  15 F  16 G  17 H  18 J  19 K  20 L  21 M  22 N  23 P  24 Q  25 R  26 S  27 T  28 U  29 V  30 W  31 X  32 Y  33 Z |