**Распознавание лиц**

С выходом версии OpenCV 2.4 появился новый класс FaceRecognizer для распознавания лиц. Текущие доступные алгоритмы:

• Eigenfaces (createEigenFaceRecognizer())

• Fisherfaces (createFisherFaceRecognizer())

• Local Binary Patterns Histogram (createLBPHFaceRecognizer())

Код распространяется по лицензии BSD.

Можно выделить следующие основные этапы процесса верификации и идентификации изображения:

• регистрация и нормализация изображения;

• выбор признаков;

• вычисление меры близости;

• построение решающего правила.

Распознавание лиц на основе геометрических характеристик лица, вероятно, наиболее интуитивно понятный подход к распознавания лиц. Одним из первых автоматизированных систем распознавания лица была описана в [Kanade73]: габаритные точки (положение глаз, ушей, носа, ...) были использованы для создания вектора признаков (расстояние между точками, угол между ними, ... ). Распознавание проводили путем расчета в евклидовой метрики. Такой метод устойчив к изменениям в освещении лица, но имеет огромный недостаток: точный учет точек маркеров является сложным, даже с современных алгоритмами. Некоторые из последних работ по геометрической распознавания лиц была проведена в [Bru92]. 22-мерный вектор признаков был использован в экспериментах на больших наборах и показал, что сами по себе геометрические особенности не несут достаточно информации для распознавания лица.

*Базы данных лиц*

Вы можете создать свою собственную базу данных или начать с одной из доступных баз данных лиц http://face-rec.org/databases/ . Три подходящих базы данных:

• AT&T Facedatabase AT&T Facedatabase, иногда также называют ORL База данных Лиц, содержит десять различных изображений, на каждую из 40 индивидуальных тематик. Для некоторых тематик, изображения были взяты в разное время, с изменением освещения, мимики (открытые / закрытые глаза, улыбаются / не улыбаются) и лицевыми деталями (очки / без очков). Все снимки были сделаны на темном однородном фоне с боковой и фронтальной позиции (с допустимы отклонение в движении).

• Yale Facedatabase, также известный как Yalefaces. AT&T Facedatabase хорошо подходит для первых испытаний, но это довольно легкая база данных. Метод главных компонент распознаёт 97% этих лиц, так что вы не увидите каких-либо серьезных улучшений с другими алгоритмами. Yale Facedatabase (также известный как Yalefaces) является более подходящим набором данных для начальных экспериментов, потому что проблема распознавания сложнее, чем кажется.База данных состоит из 15 человек (14 мужчин, 1 женщина), каждое изображений размером 320\*243 пикселей с 11 градациями серого. Есть небольшие изменения в освещение ( центральное освещение, левое освещение, правое освещение), мимике (счастливый, обычный, грустный, сонливый, удивлённый, подмигивающий) и очки (очки, без очков).

• Расширенная Yale Facedatabase B содержит 2414 изображений 38 разных людей в своей уменьшенной версии. В основе этой базы данных находится выделение признаков, которые устойчивы к освещению. Изображения в большинстве случаев устойчивы к изменениям эмоциий, окклюзий, … Этот набор данных слишком велик для экспериментов. Лучше используйте AT&T Facedatabase для своих первых тестов. Первая версия Yale Facedatabase В была использована в работе [BHK97], чтобы увидеть, как метод главных компонент и метод Fisherfaces устойчивы к изменениям освещенности. В [Lee05] использовали ту же установку для 16128 изображений 28 человек. Расширенный Yale Facedatabase B является слияние двух баз данных, которая теперь известена как расширенный Yale Facedatabase B.

*Алгоритм метода главных компонент*

Одним из наиболее известных и проработанных является метод главных компонент (principal component analysis, PCA), основанный на преобразовании Карунена-Лоева.   
 Первоначально метод главных компонент начал применяться в статистике для снижения пространства признаков без существенной потери информации. В задаче распознавания лиц его применяют главным образом для представления изображения лица вектором малой размерности (главных компонент), который сравнивается затем с эталонными векторами, заложенными в базу данных.   
 Главной целью метода главных компонент является значительное уменьшение размерности пространства признаков таким образом, чтобы оно как можно лучше описывало «типичные» образы, принадлежащие множеству лиц. Используя этот метод можно выявить различные изменчивости в обучающей выборке изображений лиц и описать эту изменчивость в базисе нескольких ортогональных векторов, которые называются собственными (eigenface).   
 Полученный один раз на обучающей выборке изображений лиц набор собственных векторов используется для кодирования всех остальных изображений лиц, которые представляются взвешенной комбинацией этих собственных векторов. Используя ограниченное количество собственных векторов можно получить сжатую аппроксимацию входному изображению лица, которую затем можно хранить в базе данных в виде вектора коэффициентов, служащего одновременно ключом поиска в базе данных лиц.   
 Вначале весь обучающий набор лиц преобразуется в одну общую матрицу данных, где каждая строка представляет собой один экземпляр изображения лица, разложенного в строку. Все лица обучающего набора должны быть приведены к одному размеру и с нормированными гистограммами.

Затем производится нормировка данных и приведение строк к 0-му среднему и 1-й дисперсии, вычисляется матрица ковариации. Для полученной матрицы ковариации решается задача определения собственных значений и соответствующих им собственных векторов (собственные лица). Далее производится сортировка собственных векторов в порядке убывания собственных значений и оставляют только первые k векторов по правилу:

> threshold (пороговое значение 0.9…0.95)

Нахождение компонентов алгоритма:

• Zero mean (среднее)

• Unit variance (единичная дисперсия)

• Covariance matrix (ковариационная матрица)

(XT- транспонированная матрица X)

• Вычисление собственных векторов

• Произведение матрицы X на k- главных компонента (k<d)

Принцип выбора базиса из первых лучших собственных векторов

Свойства главных компонент таковы, что описание объектов в пространстве k главных компонент имеет наименьшие искажения особенностей их взаимного расположения по сравнению с описанием в любом другом подпространстве той же размерности. Интерес представляет случай, когда k не велико. Тогда расположение объектов в пространстве выбранных главных компонент легко изучается визуально. При этом становится возможным делать выводы общего характера, например, выделить скопления объектов. Другая возможность использования главных компонент состоит в том, что при количественном описании объектов при проведении в дальнейшем статистического анализа ограничиваются только выделенными k компонентами (k<n). Например, в множественном регрессионном анализе вместо большого набора независимых переменных x1, x2, …, xn можно рассмотреть гораздо меньший набор главных компонент, к тому же не коррелирующих друг с другом.

Пример кода openCV С++:

#include "opencv2/core.hpp"

#include "opencv2/contrib.hpp"

#include "opencv2/highgui.hpp"

#include <iostream>

#include <fstream>

#include <sstream>

**using** **namespace** cv;

**using** **namespace** std;

**static** Mat norm\_0\_255(InputArray \_src) {

Mat src = \_src.getMat();

*// создание и нормализация изображений*

Mat dst;

**switch**(src.channels()) {

**case** 1:

cv::normalize(\_src, dst, 0, 255, NORM\_MINMAX, CV\_8UC1);

**break**;

**case** 3:

cv::normalize(\_src, dst, 0, 255, NORM\_MINMAX, CV\_8UC3);

**break**;

**default**:

src.copyTo(dst);

**break**;

}

**return** dst;

}

**static** void read\_csv(**const** string& filename, vector<Mat>& images, vector<int>& labels, char separator = ';') {

std::ifstream file(filename.c\_str(), ifstream::in);

**if** (!file) {

string error\_message = "No valid input file was given, please check the given filename.";

CV\_Error(CV\_StsBadArg, error\_message);

}

string line, path, classlabel;

**while** (getline(file, line)) {

stringstream liness(line);

getline(liness, path, separator);

getline(liness, classlabel);

**if**(!path.empty() && !classlabel.empty()) {

images.push\_back(imread(path, 0));

labels.push\_back(atoi(classlabel.c\_str()));

}

}

}

int main(int argc, **const** char \*argv[]) {

*// проверка переданных аргументов*

*// если не были переданы нужные аргументы, то создаем их (выходную папку назначаем текущую директорию)*

**if** (argc < 2) {

cout << "usage: " << argv[0] << " <csv.ext> <output\_folder> " << endl;

exit(1);

}

string output\_folder = ".";

**if** (argc == 3) {

output\_folder = string(argv[2]);

}

*// получаем путь к CSV*

string fn\_csv = string(argv[1]);

*// эти векторы данных связываются с метками labels*

vector<Mat> images;

vector<int> labels;

*// считываем данные и перехватываем исключения, если произойдет ошибка*

**try** {

read\_csv(fn\_csv, images, labels);

} **catch** (cv::Exception& e) {

cerr << "Error opening file **\"**" << fn\_csv << "**\"**. Reason: " << e.msg << endl;

*// корректное завершение*

exit(1);

}

*// выходим, если не хватает изображений*

**if**(images.size() <= 1) {

string error\_message = "This demo needs at least 2 images to work. Please add more images to your data set!";

CV\_Error(CV\_StsError, error\_message);

}

*// получаем высоту из первого изображения*

int height = images[0].rows;

*// получаем последние изображения из базы данных и тестовых изображений (еще не обученных) и удаляем их из векторов*

Mat testSample = images[images.size() - 1];

int testLabel = labels[labels.size() - 1];

images.pop\_back();

labels.pop\_back();

*// создание модели EigenFace с сохранением 10 главных компонент*

*//*

*// cv::createEigenFaceRecognizer(10);*

*//*

*// если вы хотите создать модель FaceRecognizer с пороговым значением 123.0*

*//*

*// cv::createEigenFaceRecognizer(10, 123.0);*

*//*

*// создание модели Eigenfaces с сохранением всех компонент с пороговым*

*// значением 123.0*

*//*

*// cv::createEigenFaceRecognizer(0, 123.0);*

*//*

Ptr<FaceRecognizer> model = createEigenFaceRecognizer();

model->train(images, labels);

*// распознавание по тестовым меткам*

*//*

int predictedLabel = model->predict(testSample);

*//*

*// распознавание с заданной точностью*

*//*

*// int predictedLabel = -1;*

*// double confidence = 0.0;*

*// model->predict(testSample, predictedLabel, confidence);*

*//*

string result\_message = format("Predicted class = %d / Actual class = %d.", predictedLabel, testLabel);

cout << result\_message << endl;

*// получение собственных значений eigenvalues модели Eigenfaces*

Mat eigenvalues = model->getMat("eigenvalues");

*// получение вектора собственных значений eigenvectors модели Eigenfaces*

Mat W = model->getMat("eigenvectors");

*// получение среднего из обученной модели*

Mat mean = model->getMat("mean");

*// отображение или сохранение*

**if**(argc == 2) {

imshow("mean", norm\_0\_255(mean.reshape(1, images[0].rows)));

} **else** {

imwrite(format("%s/mean.png", output\_folder.c\_str()), norm\_0\_255(mean.reshape(1, images[0].rows)));

}

*// отображение или сохранение модели EigenFace*

**for** (int i = 0; i < min(10, W.cols); i++) {

string msg = format("Eigenvalue #%d = %.5f", i, eigenvalues.at<double>(i));

cout << msg << endl;

*// получение i-ого eigenvector*

Mat ev = W.col(i).clone();

*// преобразование к исходному изображению и нормализация*

Mat grayscale = norm\_0\_255(ev.reshape(1, height));

*// показать изображение и применение Jet-colormap для лучшего восприятия*

Mat cgrayscale;

applyColorMap(grayscale, cgrayscale, COLORMAP\_JET);

*// отображение или сохранение*

**if**(argc == 2) {

imshow(format("eigenface\_%d", i), cgrayscale);

} **else** {

imwrite(format("%s/eigenface\_%d.png", output\_folder.c\_str(), i), norm\_0\_255(cgrayscale));

}

}

*// отображение или сохранение преобразованного изображения в несколько шагов*

**for**(int num\_components = min(W.cols, 10); num\_components < min(W.cols, 300); num\_components+=15) {

*// разделение eigenvectors из модели*

Mat evs = Mat(W, Range::all(), Range(0, num\_components));

Mat projection = subspaceProject(evs, mean, images[0].reshape(1,1));

Mat reconstruction = subspaceReconstruct(evs, mean, projection);

*// нормализация результата*

reconstruction = norm\_0\_255(reconstruction.reshape(1, images[0].rows));

*// отображение или сохранение*

**if**(argc == 2) {

imshow(format("eigenface\_reconstruction\_%d", num\_components), reconstruction);

} **else** {

imwrite(format("%s/eigenface\_reconstruction\_%d.png", output\_folder.c\_str(), num\_components), reconstruction);

}

}

*// отображаем, если мы не сохраняем в выходную папку*

**if**(argc == 2) {

waitKey(0);

}

**return** 0;

}

Результат работы алгоритма EigenFace с применением дополнительного фильтра для лучшего восприятия:

Если поэкспериментировать с подбором оптимального количества главных компонент:

*// отображение или сохранение преобразованных изображений в несколько шагов*

**for**(int num\_components = 10; num\_components < 300; num\_components+=15) {

*// разделение eigenvectors из модели*

Mat evs = Mat(W, Range::all(), Range(0, num\_components));

Mat projection = subspaceProject(evs, mean, images[0].reshape(1,1));

Mat reconstruction = subspaceReconstruct(evs, mean, projection);

*// нормализация результатов*

reconstruction = norm\_0\_255(reconstruction.reshape(1, images[0].rows));

*// отображение или сохранение*

**if**(argc == 2) {

imshow(format("eigenface\_reconstruction\_%d", num\_components), reconstruction);

} **else** {

imwrite(format("%s/eigenface\_reconstruction\_%d.png", output\_folder.c\_str(), num\_components), reconstruction);

}

}

Результат:

Очевидно, что 10 собственных векторов мало для хорошего преобразования. 50 собственных векторов в большинстве случаев будет достаточным для того, чтобы закодировать важные черты лица. Выбор количества собственных векторов зависит от входных данных. В [Zhao03] исследуется идеальное количество собственных векторов для данного метода.

*Алгоритм линейного дискриминанта Фишера*

Метод главных компонент хорошо зарекомендовал себя в практических приложениях. Однако, в тех случаях, когда на изображении лица присутствуют значительные изменения в освещенности или выражении лица, эффективность метода значительно падает. Все дело в том, что PCA выбирает подпространство с такой целью, чтобы максимально аппроксимировать входной набор данных, а не выполнить дискриминацию между классами лиц.   
 Тогда было предложено решение этой проблемы с использование линейного дискриминанта Фишера (в литературе встречается название “Eigen-Fisher”, “Fisherface”, LDA). LDA выбирает линейное подпространство, которое максимизирует отношение.

Пусть X случайный вектор с образцами, взятых из С-классы:

Разброс матрицы SB и SW рассчитывается как:

, где общая средняя

,

а

  Ni - число представителей в i-м классе, N- общее число представителей в обучающей выборке, xij - j-й представитель i-го класса.

Классический алгоритм Фишера теперь ищет проекции W, которые максимизирует отдельные классы критериев:

Решение этой задачи оптимизации задается решением общей задачи на собственные значения:

В задачах распознавания образов количество образцов N почти всегда меньше, чем размер входных данных (количество пикселей), так разброс матрица SW становится сингулярной. Эта задача была решена путем выполнения главных компонент по данным и проектирование образцов в n-мерного пространства.

Задача оптимизации может быть переписана в виде:

Матрица преобразования W c проекционными образцами в (с-1) -мерном пространстве можно получить по формуле:

Собственный вектор соответствует наибольшему собственному значению матрицы W.

Пример кода openCV С++:

#include "opencv2/core.hpp"

#include "opencv2/contrib.hpp"

#include "opencv2/highgui.hpp"

#include <iostream>

#include <fstream>

#include <sstream>

**using** **namespace** cv;

**using** **namespace** std;

**static** Mat norm\_0\_255(InputArray \_src) {

Mat src = \_src.getMat();

*// создание и нормализация изображений*

Mat dst;

**switch**(src.channels()) {

**case** 1:

cv::normalize(\_src, dst, 0, 255, NORM\_MINMAX, CV\_8UC1);

**break**;

**case** 3:

cv::normalize(\_src, dst, 0, 255, NORM\_MINMAX, CV\_8UC3);

**break**;

**default**:

src.copyTo(dst);

**break**;

}

**return** dst;

}

**static** void read\_csv(**const** string& filename, vector<Mat>& images, vector<int>& labels, char separator = ';') {

std::ifstream file(filename.c\_str(), ifstream::in);

**if** (!file) {

string error\_message = "No valid input file was given, please check the given filename.";

CV\_Error(CV\_StsBadArg, error\_message);

}

string line, path, classlabel;

**while** (getline(file, line)) {

stringstream liness(line);

getline(liness, path, separator);

getline(liness, classlabel);

**if**(!path.empty() && !classlabel.empty()) {

images.push\_back(imread(path, 0));

labels.push\_back(atoi(classlabel.c\_str()));

}

}

}

int main(int argc, **const** char \*argv[]) {

*// проверка переданных аргументов*

*// если не были переданы нужные аргументы, то создаем их (выходную папку назначаем текущую директорию)*

**if** (argc < 2) {

cout << "usage: " << argv[0] << " <csv.ext> <output\_folder> " << endl;

exit(1);

}

string output\_folder = ".";

**if** (argc == 3) {

output\_folder = string(argv[2]);

}

*// получаем путь к CSV*

string fn\_csv = string(argv[1]);

*// эти векторы данных связываются с метками labels*

vector<Mat> images;

vector<int> labels;

*// считываем данные и перехватываем исключения, если произойдет ошибка*

**try** {

read\_csv(fn\_csv, images, labels);

} **catch** (cv::Exception& e) {

cerr << "Error opening file **\"**" << fn\_csv << "**\"**. Reason: " << e.msg << endl;

*// корректное завершение*

exit(1);

}

*// выходим, если не хватает изображений*

**if**(images.size() <= 1) {

string error\_message = "This demo needs at least 2 images to work. Please add more images to your data set!";

CV\_Error(CV\_StsError, error\_message);

}

*// получаем высоту из первого изображения*

int height = images[0].rows;

*// получаем последние изображения из базы данных и тестовых изображений (еще не обученных) и удаляем их из векторов*

Mat testSample = images[images.size() - 1];

int testLabel = labels[labels.size() - 1];

images.pop\_back();

labels.pop\_back();

*// создание модели EigenFace с сохранением 10 главными компонентами*

*//*

*// cv::createEigenFaceRecognizer(10);*

*//*

*// если вы хотите создать модель FaceRecognizer с пороговым значением 123.0*

*//*

*// cv::createEigenFaceRecognizer(10, 123.0);*

*//*

*// создание модели Eigenfaces с сохранением всех компонент с пороговым*

*// значением 123.0*

*//*

*// cv::createEigenFaceRecognizer(0, 123.0);*

*//*

Ptr<FaceRecognizer> model = createFisherFaceRecognizer();

model->train(images, labels);

*// распознование по тестовым меткам*

int predictedLabel = model->predict(testSample);

*//*

*// распознавание с заданной точностью*

*//*

*// int predictedLabel = -1;*

*// double confidence = 0.0;*

*// model->predict(testSample, predictedLabel, confidence);*

*//*

string result\_message = format("Predicted class = %d / Actual class = %d.", predictedLabel, testLabel);

cout << result\_message << endl;

*// получение собственных значений eigenvalues модели Eigenfaces*

Mat eigenvalues = model->getMat("eigenvalues");

*// получение собственных значений eigenvalues модели Eigenfaces*

Mat W = model->getMat("eigenvectors");

*// получение среднего из обученной модели*

Mat mean = model->getMat("mean");

*// отображение или сохранение*

**if**(argc == 2) {

imshow("mean", norm\_0\_255(mean.reshape(1, images[0].rows)));

} **else** {

imwrite(format("%s/mean.png", output\_folder.c\_str()), norm\_0\_255(mean.reshape(1, images[0].rows)));

}

*// отображение или сохранение первого; и не более чем 16*

**for** (int i = 0; i < min(16, W.cols); i++) {

string msg = format("Eigenvalue #%d = %.5f", i, eigenvalues.at<double>(i));

cout << msg << endl;

*// получение i-ого eigenvector*

Mat ev = W.col(i).clone();

*// преобразование к исходному изображению и нормализация*

Mat grayscale = norm\_0\_255(ev.reshape(1, height));

*// показать изображение и применение Bone-colormap для лучшего восприятия*

Mat cgrayscale;

applyColorMap(grayscale, cgrayscale, COLORMAP\_BONE);

*// отображение или сохранение*

**if**(argc == 2) {

imshow(format("fisherface\_%d", i), cgrayscale);

} **else** {

imwrite(format("%s/fisherface\_%d.png", output\_folder.c\_str(), i), norm\_0\_255(cgrayscale));

}

}

*// отображение или сохранение преобразованного изображения в несколько шагов*

**for**(int num\_component = 0; num\_component < min(16, W.cols); num\_component++) {

*// разделение eigenvectors из модели*

Mat ev = W.col(num\_component);

Mat projection = subspaceProject(ev, mean, images[0].reshape(1,1));

Mat reconstruction = subspaceReconstruct(ev, mean, projection);

*// нормализация результата*

reconstruction = norm\_0\_255(reconstruction.reshape(1, images[0].rows));

*// отображение или сохранение*

**if**(argc == 2) {

imshow(format("fisherface\_reconstruction\_%d", num\_component), reconstruction);

} **else** {

imwrite(format("%s/fisherface\_reconstruction\_%d.png", output\_folder.c\_str(), num\_component), reconstruction);

}

}

*// отображаем, если мы не сохраняем в выходную папку*

**if**(argc == 2) {

waitKey(0);

}

**return** 0;

}

Результат работы алгоритма FisherFace с использованием базы данных Yale Facedatabase A и применением фильтра для результата:

Другой результат:

*Алгоритм локальных бинарных шаблонов*

Локальный бинарный шаблон – это определенный вид признака, используемый для классификации в компьютерном зрении, и представляющий собой простой оператор. Оператор LBP может быть использован для поиска объекта на изображении (например лица), а также проверки этого объекта на принадлежность некоторому классу (верификация, распознавание эмоций, пола по лицу). Оператор LBP впервые был предложен T.Ojala в 1996 году. Он представляет собой эффективный оператор, который представляет каждый пиксель изображения в виде бинарного числа, зависящего от интенсивностей соседних пикселей изображения.   
 Этот оператор является эффективным в вычислительном плане, так как работает только с целочисленной арифметикой (это позволяет достигать real-time производительности в некоторых задачах), а также он инвариантен к изменениям яркости на изображении, вызванным съемкой в различных условиях освещения.  
 Классический LBP, который применяется к пикселю изображения, использует восемь пикселей окрестности.

Он принимает центральный пиксель в качестве порога и сравнивает значение яркости в каждом пикселе окрестности с ним. Если это значение больше порога (или равное значение), то пиксель принимает значение 1. Если же меньше - 0. Полученное восьмибитное число характеризует окрестность пикселя. Всего вариантов таких чисел 28=256. Таким образом каждому пикселю изображения присваивается одна из 256 меток, характеризующих его. Далее из этих данных не составляет труда построить гистограмму и сравнивать текстуры по гистограммам LBP.

Однако хотелось бы использовать окрестности произвольного радиуса с произвольным числом значащих пикселей. В 2002 году был предложен метод eLBP (extended LBP), который удовлетворяет этим условиям. Использование круговой окрестности и билинейной интерполяции значений интенсивностей пикселей позволяет построить локальный бинарный шаблон с произвольным количеством точек P и радиусом R:

Некоторые бинарные коды несут в себе больше информации, чем остальные. Так, локальный бинарный шаблон называется равномерным, если он содержит не более трех серий «0» и «1» (например, 00000000, 001110000 и 11100001). Во-первых, равномерные LBP определяют только важные локальные особенности изображения, такие как концы линий, грани, углы и пятна. Во-вторых, они обеспечивают существенную экономию памяти различных шаблонов вместо .

Более формальное описание оператора LBP можно задать в виде:

,

c в качестве центрального пикселя с интенсивностью ; - интенсивность соседнего пикселя; - является функцией и определяется как:

Это описание позволяет захватить очень маленькие детали в изображениях.

Теперь получим оператор eLBP:

Для заданной точки (xC, yC) положение соседа (xP, yP), может быть рассчитано по формуле:

Где R является радиус окружности и P является число точек отбора.

Применяя оператор LBP к каждому пикселю изображения, мы можем построить гистограмму, в которой каждому равномерному коду LBP соответствует отдельный столбец. Также имеется еще один дополнительный столбец, который содержит информацию обо всех неравномерных шаблонах.  
 Изображения лиц могут рассматриваться как набор всевозможных локальных особенностей, которые хорошо описываются с помощью локальных бинарных шаблонов. Однако гистограмма, построенная для всего изображения в целом, кодирует лишь наличие тех или иных локальных особенностей, но при этом не содержит никакой информации об их расположении на изображении. Для учета такого рода информации изображение разбивается на подобласти, в каждой из которых вычисляется своя гистограмма LBP. Путем конкатенации этих гистограмм может быть получена общая гистограмма, учитывающая как локальные, так и глобальные особенности изображения. При таком подходе для лучшего извлечения признаков можно варьировать параметры оператора LBP и число разбиений изображения на подобласти.

Пример кода openCV С++:

#include "opencv2/core.hpp"

#include "opencv2/contrib.hpp"

#include "opencv2/highgui.hpp"

#include <iostream>

#include <fstream>

#include <sstream>

**using** **namespace** cv;

**using** **namespace** std;

**static** void read\_csv(**const** string& filename, vector<Mat>& images, vector<int>& labels, char separator = ';') {

std::ifstream file(filename.c\_str(), ifstream::in);

**if** (!file) {

string error\_message = "No valid input file was given, please check the given filename.";

CV\_Error(CV\_StsBadArg, error\_message);

}

string line, path, classlabel;

**while** (getline(file, line)) {

stringstream liness(line);

getline(liness, path, separator);

getline(liness, classlabel);

**if**(!path.empty() && !classlabel.empty()) {

images.push\_back(imread(path, 0));

labels.push\_back(atoi(classlabel.c\_str()));

}

}

}

int main(int argc, **const** char \*argv[]) {

*// проверка переданных аргументов*

*// если не были переданы нужные аргументы, то выходим*

**if** (argc != 2) {

cout << "usage: " << argv[0] << " <csv.ext>" << endl;

exit(1);

}

*// получаем путь к CSV*

string fn\_csv = string(argv[1]);

*// эти векторы данных связываются с метками labels*

vector<Mat> images;

vector<int> labels;

*// считываем данные и перехватываем исключения, если произойдет ошибка*

**try** {

read\_csv(fn\_csv, images, labels);

} **catch** (cv::Exception& e) {

cerr << "Error opening file **\"**" << fn\_csv << "**\"**. Reason: " << e.msg << endl;

*// корректное завершение*

exit(1);

}

*// выходим, если не хватает изображений*

**if**(images.size() <= 1) {

string error\_message = "This demo needs at least 2 images to work. Please add more images to your data set!";

CV\_Error(CV\_StsError, error\_message);

}

*// получаем высоту из первого изображения*

int height = images[0].rows;

*// получаем последние изображения из базы данных и тестовых изображений (еще не обученных) и удаляем их из векторов*

Mat testSample = images[images.size() - 1];

int testLabel = labels[labels.size() - 1];

images.pop\_back();

labels.pop\_back();

*// создание LBPH модель*

*//*

*// значения по умолчанию*

*//*

*// radius = 1*

*// neighbors = 8*

*// grid\_x = 8*

*// grid\_y = 8*

*//*

*// если вы хотите модель LBPH FaceRecognizer с radius=2 и neighbors=16*

*//*

*// cv::createLBPHFaceRecognizer(2, 16);*

*//*

*// если вы хотите использовать значения по умолчанию и добавить порговое значение 123.0*

*//*

*// cv::createLBPHFaceRecognizer(1,8,8,8,123.0)*

*//*

Ptr<FaceRecognizer> model = createLBPHFaceRecognizer();

model->train(images, labels);

*// распознавание по тестовым меткам*

int predictedLabel = model->predict(testSample);

*//*

*// распознавание с заданной точностью*

*//*

*// int predictedLabel = -1;*

*// double confidence = 0.0;*

*// model->predict(testSample, predictedLabel, confidence);*

*//*

string result\_message = format("Predicted class = %d / Actual class = %d.", predictedLabel, testLabel);

cout << result\_message << endl;

*// изменение порогового значения модели и повторное распознавание*

*//*

model->set("threshold", 0.0);

predictedLabel = model->predict(testSample);

cout << "Predicted class = " << predictedLabel << endl;

*// получение информации о модели и вывод ее на экран*

cout << "Model Information:" << endl;

string model\_info = format("**\t**LBPH(radius=%i, neighbors=%i, grid\_x=%i, grid\_y=%i, threshold=%.2f)",

model->getInt("radius"),

model->getInt("neighbors"),

model->getInt("grid\_x"),

model->getInt("grid\_y"),

model->getDouble("threshold"));

cout << model\_info << endl;

*// мы можем получить гистограммы и вывести на экран*

vector<Mat> histograms = model->getMatVector("histograms");

cout << "Size of the histograms: " << histograms[0].total() << endl;

**return** 0;

}

Результат работы программы:

Литература

[AHP04]

*(1, 2, 3)* Ahonen, T., Hadid, A., and Pietikainen, M. *Face Recognition with Local Binary Patterns.* Computer Vision - ECCV 2004 (2004), 469–481.

[BHK97]

*(1, 2, 3, 4, 5)* Belhumeur, P. N., Hespanha, J., and Kriegman, D. *Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection.* IEEE Transactions on Pattern Analysis and Machine Intelligence 19, 7 (1997), 711–720.

[Bru92]

Brunelli, R., Poggio, T. *Face Recognition through Geometrical Features.* European Conference on Computer Vision (ECCV) 1992, S. 792–800.

[Duda01]

Duda, Richard O. and Hart, Peter E. and Stork, David G., *Pattern Classification* (2nd Edition) 2001.

[Fisher36]

Fisher, R. A. *The use of multiple measurements in taxonomic problems.* Annals Eugen. 7 (1936), 179–188.

[GBK01]

Georghiades, A.S. and Belhumeur, P.N. and Kriegman, D.J., *From Few to Many: Illumination Cone Models for Face Recognition under Variable Lighting and Pose* IEEE Transactions on Pattern Analysis and Machine Intelligence 23, 6 (2001), 643-660.

[Kanade73]

Kanade, T. *Picture processing system by computer complex and recognition of human faces.* PhD thesis, Kyoto University, November 1973

[KM01]

Martinez, A and Kak, A. *PCA versus LDA* IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 23, No.2, pp. 228-233, 2001.

[Lee05]

Lee, K., Ho, J., Kriegman, D. *Acquiring Linear Subspaces for Face Recognition under Variable Lighting.* In: IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 27 (2005), Nr. 5

[Messer06]

Messer, K. et al. *Performance Characterisation of Face Recognition Algorithms and Their Sensitivity to Severe Illumination Changes.* In: In: ICB, 2006, S. 1–11.

[RJ91]

• Raudys and A.K. Jain. *Small sample size effects in statistical pattern recognition: Recommendations for practitioneers.* - IEEE Transactions on Pattern Analysis and Machine Intelligence 13, 3 (1991), 252-264.

[Tan10]

Tan, X., and Triggs, B. *Enhanced local texture feature sets for face recognition under difficult lighting conditions.* IEEE Transactions on Image Processing 19 (2010), 1635–650.

[TP91]

Turk, M., and Pentland, A. *Eigenfaces for recognition.* Journal of Cognitive Neuroscience 3 (1991), 71–86.

[Tu06]

Chiara Turati, Viola Macchi Cassia, F. S., and Leo, I. *Newborns face recognition: Role of inner and outer facial features. Child Development* 77, 2 (2006), 297–311.

[Wiskott97]

Wiskott, L., Fellous, J., Krüger, N., Malsburg, C. *Face Recognition By Elastic Bunch Graph Matching.* IEEE Transactions on Pattern Analysis and Machine Intelligence 19 (1997), S. 775–779

[Zhao03]

Zhao, W., Chellappa, R., Phillips, P., and Rosenfeld, A. Face recognition: A literature survey. ACM Computing Surveys (CSUR) 35, 4 (2003), 399–458.