Санкт-Петербургский Государственный Политехнический Университет

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Факультет Технической Кибернетики  
Кафедра Информационных и Управляющих Систем

**Отчет по курсовому проекту по дисциплине**

«Современные проблемы информатики и вычислительной техники»

**Тема:**

«Распознавание и воспроизведение на виртуальной модели   
эмоций человека на основе анализа мимики.   
Подсистема распознавания мимических движений»

Выполнил: В.С. Бут

6084/10

Проверил: В.П. Шкодырев

Санкт-Петербург

2010

Оглавление

[Постановка задачи 2](#_Toc281192605)

[Анализ предметной области 2](#_Toc281192606)

[База образцов входных воздействий 2](#_Toc281192607)

[Метод преобразования входных воздействий в вектор признаков 3](#_Toc281192608)

[Анализ основных компонентов 3](#_Toc281192609)

[Признаки Хаара 3](#_Toc281192610)

[Модель компьютерного обучения 4](#_Toc281192611)

[Выбор параметров модели 5](#_Toc281192612)

[Реализация 5](#_Toc281192613)

[Определение координат лица 6](#_Toc281192614)

[Построение вектора признаков 6](#_Toc281192615)

[Классификация эмоционального состояния 6](#_Toc281192616)

[Список литературы 7](#_Toc281192617)

[Приложение 1. Основной исходный код ядра распознавания эмоций (модуль emocore) 8](#_Toc281192618)

[Приложение 2. Дополнительные средства 14](#_Toc281192619)

[Модуль emoextract 14](#_Toc281192620)

[Модуль emoeigen 20](#_Toc281192621)

[Модуль emoshow 21](#_Toc281192622)

[Модуль emotrain 22](#_Toc281192623)

# Постановка задачи

Разработать систему распознавания эмоций человека и отображения их на виртуальной компьютерной модели.

Распознавание эмоций производится на основе анализа мимики человека, находящегося фронтально перед цифровой видеокамерой. Алгоритм распознавания эмоций либо выбирается из числа удовлетворяющих условиям настоящего технического задания описанных в литературе, либо разрабатывается самостоятельно. Распознаванию подлежат базовые психоэмоциональные состояния человека и их физические проявления (радость, грусть, смех, плач и др.) в режиме реального времени (время распознавания t ≤ 2 сек). Для повышения точности работы необходимо провести предварительный анализ психосоматических реакций, которые могут быть обнаружены визуально.

Работа системы может быть реализована как с использованием механизма предварительного обучения, так и без него. В случае разработки системы без обучения, необходимо обеспечить её работу в виде распознавания мимики, соответствующей эмоциональным состояниям радости, печали, гнева, удивления для любого объекта анализа. В случае использования механизма предварительного обучения система использует предварительные данные о физических проявлениях базовых эмоций радости, удивления, печали, гнева, отвращения, страха, интереса, презрения определенного объекта.

# Анализ предметной области

Задача распознавания эмоций относится к классу задач распознавания и классификации образов. Такие задачи эффективнее всего решаются с помощью методов компьютерного обучения.

Принцип методов компьютерного обучения заключается в том, что любому входному воздействию ставится в соответствие некоторый вектор признаков. Затем, на основе набора образцовых входных воздействий, преобразованных в векторы признаков, производится обучение некоторой модели. В результате, обученная модель по любому входному воздействию в виде вектора признаков выдает результат, основанный на заложенной во время обучения информации.

Таким образом, для построения требуемой системы необходимо решить следующие задачи:

1. подбор базы образцов входных воздействий;
2. выбор метода преобразования входных воздействий в вектор признаков;
3. выбор модели компьютерного обучения;
4. выбор параметров модели компьютерного обучения;
5. реализация метода преобразования входных воздействий в вектор признаков и модели обучения.

# База образцов входных воздействий

В рамках нашей задачи входными воздействиями являются изображения лиц с различными эмоциональными состояниями. Таким образом, необходимо было собрать большое число фотографий людей с различными эмоциями.

База собиралась из следующих источников:

1. Фотографии, сделанные самостоятельно на цифровые фотокамеры;
2. Фотографии из открытых фотоальбомов пользователей социальных сетей;
3. Фотографии из открытых баз данных:
   1. Jaffe (http://www.kasrl.org/jaffe.html)
   2. Yale Faces Database (http://cvc.yale.edu/projects/yalefaces/yalefaces.html)

Поскольку собранная база фотографий имела относительно небольшой размер, был написан дополнительный инструмент (emoextract), позволяющий генерировать на основе одного изображения серию образцов, незначительно отличающихся друг от друга.

# Метод преобразования входных воздействий в вектор признаков

Входными воздействиями для нашей системы являются изображения. Следовательно, необходимо использовать методы выделения признаков, применимые к двумерным изображениям. Наиболее подходящими методами для этой задачи являются:

* метод «Анализа основных компонентов»;
* метод «признаков Хаара».

Эти методы были выбраны потому, что они обеспечивают выделение признаков, слабо зависящих от относительных размеров изображений.

## Анализ основных компонентов

Суть этого метода заключается в выделении из исходного множества изображений некоторого подмножества «основных компонентов», такого что:

* мощность сходного множества изображений больше мощности подмножества «основных компонентов»;
* любое изображение из исходного множества представимо в виде линейной комбинации «основных компонентов».

Таким образом, используя представление изображений в виде линейной комбинации, вектор признаков формируется из коэффициентов этой линейной комбинации.

## Признаки Хаара

Метод признаков Хаара основан на применении к изображению ряда фильтров. Фильтры Хаара применяются к изображению, а из значений этих фильтров формируется вектор признаков.

Ниже изображены используемые фильтры Хаара. Значение фильтра определяется разностью двух сумм пикселей, попавших соответственно в белую и черную области. Эта разность умножается на коэффициент, обратный площади фильтра, для того, чтобы признаки не зависели от размеров изображения.



Рис. Фильтры Хаара

# Модель компьютерного обучения

Для компьютерного обучения была выбрана модель многослойного персептрона (Multi-Layer Perceptron, MLP), поскольку она обладает большим быстродействием, а также показывает хорошие результаты при относительно небольших объемах обучающей информации.

Сама модель многослойного персептрона может быть изображена следующим образом:



Рис. Модель многослойного персептрона

где каждый элемент представляет собой персептрон. Персептрон – это математическая модель нейрона, которая описывается набором весовых коэффициентов на входах, смещением входного воздействия, а также функцией активации, нелинейной в подавляющем большинстве случаев.

В самом простом примере функция активации представляет собой «ступеньку», однако более удобным для математического анализа и обучения является функция активации в виде «симметричного сигмоида».



Рис. Персептрон – математическая модель нейрона



Рис. Симметричный сигмоид

# Выбор параметров модели

Выбор параметров модели производился в процессе обучения и тестирования. Основными параметрами модели многослойного персептрона являются число и размеры слоев.

Число слоев было выбрано равным трем, поскольку оно обеспечивает распознавание классов объектов, описываемых невыпуклыми множествами.

Размеры первого и последнего слоев совпадают соответственно с числом признаков, выделяемых из входных изображений, и с числом распознаваемых эмоций. Размер среднего или, как его называют, скрытого слоя устанавливается экспериментально.

При подборе параметров модели существуют 2 правила, которые позволяют определить нужное направление изменения параметров [2]:

1. Если обученная модель хорошо распознает те изображения, на которых производилось обучение, а другие изображения она распознает плохо, это говорит о том, что размер модели слишком большой. То есть модель кроме «полезных» закономерностей запоминает и «шумы» во входных образцах. Следовательно, требуется уменьшить размер модели.
2. Если модель одинаково плохо распознает как изображения, на которых производилось обучение, так и другие изображения, следовательно, размер модели недостаточно велик.

# Реализация

Реализация методов производилась на основе библиотеки Open Computer Vision Library (OpenCV, <http://opencv.willowgarage.com/wiki/>).

Для построения ядра системы распознавания, была реализована следующая последовательность вызовов:

* определение координат лица на входном изображении;
* построение вектора признаков для изображения лица;
* классификация эмоционального состояния по вектору признаков.

## Определение координат лица

Для определения координат лица был использован каскадный классификатор Хаара. Он представляет собой каскад решающих деревьев, обученный для классификации изображений по признаку «лицо – не лицо». На вход каскада решающих деревьев поступает вектор признаков, состоящий из значений фильтров Хаара.

Библиотека OpenCV содержит реализацию этого классификатора, а также заранее обученную модель, которую можно загружать из подготовленного xml-файла.

## Построение вектора признаков

Для построения вектора признаков было реализовано 2 метода: метод «Анализа основных компонентов» и метод «признаков Хаара».

При тестировании системы с использованием данных методов (при прочих равных условиях) были получены следующие результаты:

* метод «Анализа основных компонентов» показал хорошее качество распознавания эмоций, когда на вход системы подавались качественные изображения лиц с малым уровнем шумов;
* метод «признаков Хаара» показал более хорошие результаты на сильно-зашумленных изображениях (например, на таких изображениях, которые можно получить с помощью современных веб-камер).

Метод «Анализа основных компонентов» реализован в библиотеке OpenCV в виде отдельного класса. Для работы с этим классом в ядро системы были добавлены функции сохранения данных метода в xml-файл на жесткий диск и загрузки их из xml-файла.

Метод «признаков Хаара» не имеет доступной реализации в библиотеке OpenCV и был полностью реализован в ядре системы.

## Классификация эмоционального состояния

Для классификации эмоционального состояния используется реализация многослойного персептрона из библиотеки OpenCV. Обучение этой модели производится с помощью дополнительного модуля emotrain.

# Список литературы

1. Леонтьев В.О. Классификация эмоций. Издательство Инновационно-ипотечного центра, Одесса, 2002.
2. Gary Bradski and Adrian Kaehler, Learning OpenCV, O’Reilly Media, 2008.
3. Gregory Shakhnarovich, Baback Moghaddam, Face Recognition in Subspaces, Mitsubishi Electric Research Laboratories, Inc., 2004.
4. Mukesh T. Patel and D. C. Joshi, Human Hidden Emotion Identification Techniques with Multi-Feature Face Recognition System, International Journal of Computational and Applied Mathematics, ISSN 1819-4966 Volume 4 Number 1 (2009), pp. 47‑57.
5. OpenCV Reference Manual, v2.1, 2010.

# Приложение 1. Основной исходный код ядра распознавания эмоций (модуль emocore)

Основной интерфейс взаимодействия графической оболочки системы с ядром:

class EmoCore {

public:

static shared\_ptr<EmoCore> create();

protected:

EmoCore();

public:

virtual ~EmoCore();

static std::string errorMessage(int errcode);

virtual int init(std::vector<unsigned char> &classes,

const std::map<std::string, std::string> &parameters) = 0;

virtual int extractFace(const cv::Mat &img, cv::Rect &face) = 0;

virtual int guess(const cv::Mat &face,

std::map<unsigned char, float> &results) = 0;

};

Реализация интерфейса взаимодействия:

class EmoCoreImpl : public EmoCore {

public:

EmoCoreImpl();

int init(std::vector<unsigned char> &classes,

const std::map<std::string, std::string> &parameters);

int extractFace(const cv::Mat &img, cv::Rect &face);

int guess(const cv::Mat &face, std::map<unsigned char, float> &results);

const std::vector<unsigned char> &classes() const;

private:

void convertFace(const cv::Mat &face, cv::Mat &converted) const;

private:

bool myInitialized;

std::vector<unsigned char> myClasses;

cv::CascadeClassifier myCvCascade;

cv::PCA myCvPCA;

CvANN\_MLP myCvMLP;

float myScale;

float myMinFace;

int myFaceSize;

};

Метод нахождения координат лица на изображении:

int EmoCoreImpl::extractFace(const cv::Mat &img, cv::Rect &face) {

if (!myInitialized) {

return EMOERR\_NOT\_INITIALIZED;

}

cv::Mat buffer;

cv::cvtColor(img, buffer, CV\_BGR2GRAY);

cv::equalizeHist(buffer, buffer);

std::vector<cv::Rect> faces;

myCvCascade.detectMultiScale(buffer, faces, myScale, 2,

CV\_HAAR\_SCALE\_IMAGE | CV\_HAAR\_FIND\_BIGGEST\_OBJECT,

cv::Size(buffer.cols \* myMinFace, buffer.rows \* myMinFace));

if (faces.empty()) {

face = cv::Rect(0, 0, 0, 0);

} else {

face = faces[0];

}

return 0;

}

Метод определения вектора признаков (с использованием метода «Анализа основных компонентов») и классификации изображения на основе этого вектора:

int EmoCoreImpl::guess(const cv::Mat &face,

std::map<unsigned char, float> &results) {

if (!myInitialized) {

return EMOERR\_NOT\_INITIALIZED;

}

cv::Mat converted;

convertFace(face, converted);

cv::Mat features;

myCvPCA.project(converted.reshape(0, 1), features);

cv::Mat output;

myCvMLP.predict(features, output);

float maxValue = 0.0f;

for (int j = 0; j < output.cols; ++j) {

float &prob = output.at<float>(0, j);

if (prob < 0.0f) {

prob = 0.0f;

} else if (prob > 1.0f) {

maxValue = prob;

}

}

if (maxValue > 1.0f) {

std::cerr << "maxValue = " << maxValue << std::endl;

output /= maxValue;

}

results.clear();

for (int j = 0; j < output.cols; ++j) {

const float prob = output.at<float>(0, j);

const unsigned char letter = myClasses[j];

results[letter] = prob;

}

return 0;

}

Дополнительный метод начального преобразования изображений:

void EmoCoreImpl::convertFace(const cv::Mat &face, cv::Mat &converted) const {

cv::Mat gray, resized;

cv::cvtColor(face, gray, CV\_BGR2GRAY);

cv::resize(gray, resized, cv::Size(myFaceSize, myFaceSize),

0, 0, cv::INTER\_LINEAR);

cv::equalizeHist(resized, resized);

resized.convertTo(converted, CV\_32FC1, 1.0 / 255);

}

Класс реализации метода извлечения признаков на основе фильтров Хаара:

#define EMO\_FEATURES\_MAX\_RECS 2

class EmoFeatures {

struct Feature {

int Width;

int Height;

struct FeatureRect {

cv::Rect Rect;

float Weight;

} Recs[EMO\_FEATURES\_MAX\_RECS];

Feature();

Feature(int w, int h, const cv::Rect &rect0, float weight0,

const cv::Rect &rect1, float weight1);

float compute(const cv::Mat &integral, int x, int y) const;

float computeTilted(const cv::Mat &integral, int x, int y) const;

};

public:

EmoFeatures(int size);

void project(const cv::Mat &image, cv::Mat &features);

int featuresNumber(const cv::Size &imageSize);

private:

const int mySize;

std::vector<Feature> myFeatures;

};

EmoFeatures::Feature::Feature() {

Width = 0;

Height = 0;

for (int i = 0; i < EMO\_FEATURES\_MAX\_RECS; ++i) {

FeatureRect &r = Recs[i];

r.Rect = cv::Rect(0, 0, 0, 0);

r.Weight = 0.f;

}

}

EmoFeatures::Feature::Feature(int w, int h, const cv::Rect &rect0,

float weight0, const cv::Rect &rect1, float weight1) {

Width = w;

Height = h;

for (int i = 0; i < EMO\_FEATURES\_MAX\_RECS; ++i) {

FeatureRect &r = Recs[i];

r.Rect = cv::Rect(0, 0, 0, 0);

r.Weight = 0.f;

}

Recs[0].Rect = rect0;

Recs[0].Weight = weight0;

Recs[1].Rect = rect1;

Recs[1].Weight = weight1;

}

float EmoFeatures::Feature::compute(const cv::Mat &integral,

int x, int y) const {

float res = 0.f;

for (int i = 0; i < EMO\_FEATURES\_MAX\_RECS; ++i) {

const FeatureRect &r = Recs[i];

if (r.Weight != 0.f) {

const int x0 = x + r.Rect.x;

const int y0 = y + r.Rect.y;

const int x1 = x0 + r.Rect.width;

const int y1 = y0 + r.Rect.height;

res += r.Weight \* (integral.at<double>(y1, x1)

- integral.at<double>(y1, x0)

- integral.at<double>(y0, x1)

+ integral.at<double>(y0, x0));

}

}

return res;

}

float EmoFeatures::Feature::computeTilted(const cv::Mat &integral,

int x, int y) const {

float res = 0.f;

for (int i = 0; i < EMO\_FEATURES\_MAX\_RECS; ++i) {

const FeatureRect &r = Recs[i];

if (r.Weight != 0.f) {

const int x0 = x + r.Rect.x - r.Rect.y;

const int y0 = y + r.Rect.x + r.Rect.y;

const int x1 = x0 + r.Rect.width;

const int y1 = y0 + r.Rect.width;

const int x2 = x0 - r.Rect.height;

const int y2 = y0 + r.Rect.height;

const int x3 = x2 + r.Rect.width;

const int y3 = y2 + r.Rect.width;

res += r.Weight \* (integral.at<double>(y3, x3)

- integral.at<double>(y2, x2)

- integral.at<double>(y1, x1)

+ integral.at<double>(y0, x0));

}

}

return res;

}

EmoFeatures::EmoFeatures(int size) : mySize(size) {

myFeatures.assign(7, Feature());

float sz2 = 1.f / (size \* size);

myFeatures[0] = Feature(4 \* size, 4 \* size,

cv::Rect(0, 0, 2 \* size, 4 \* size), 1.f \* sz2,

cv::Rect(2, 0, 2 \* size, 4 \* size), -1.f \* sz2);

myFeatures[1] = Feature(4 \* size, 4 \* size,

cv::Rect(0, 0, 4 \* size, 2 \* size), 1.f \* sz2,

cv::Rect(0, 2, 4 \* size, 2 \* size), -1.f \* sz2);

myFeatures[2] = Feature(3 \* size, 2 \* size,

cv::Rect(0, 0, 3 \* size, 2 \* size), 1.f \* sz2,

cv::Rect(1, 0, 1 \* size, 2 \* size), -3.f \* sz2);

myFeatures[3] = Feature(4 \* size, 2 \* size,

cv::Rect(0, 0, 4 \* size, 2 \* size), 1.f \* sz2,

cv::Rect(1, 0, 2 \* size, 2 \* size), -2.f \* sz2);

myFeatures[4] = Feature(2 \* size, 3 \* size,

cv::Rect(0, 0, 2 \* size, 3 \* size), 1.f \* sz2,

cv::Rect(0, 1, 2 \* size, 1 \* size), -3.f \* sz2);

myFeatures[5] = Feature(2 \* size, 4 \* size,

cv::Rect(0, 0, 2 \* size, 4 \* size), 1.f \* sz2,

cv::Rect(0, 1, 2 \* size, 2 \* size), -2.f \* sz2);

myFeatures[6] = Feature(3 \* size, 3 \* size,

cv::Rect(0, 0, 3 \* size, 3 \* size), 1.f \* sz2,

cv::Rect(1, 1, 1 \* size, 1 \* size), -9.f \* sz2);

}

void EmoFeatures::project(const cv::Mat &image, cv::Mat &features) {

int count = featuresNumber(image.size());

if (features.rows != 1 || features.cols != count

|| features.type() != CV\_32FC1) {

features.create(1, count, CV\_32FC1);

}

cv::Mat sum, sqsum, tilted;

cv::integral(image, sum, sqsum, tilted, CV\_64F);

int featureNumber = 0;

for (size\_t i = 0; i < myFeatures.size(); ++i) {

const Feature &f = myFeatures[i];

for (int y = 0; y + f.Height < sum.rows; y += f.Height) {

for (int x = 0; x + f.Width < sum.cols; x += f.Width) {

features.at<float>(0, featureNumber++) =

f.compute(sum, x, y);

}

}

}

for (size\_t i = 0; i < myFeatures.size(); ++i) {

const Feature &f = myFeatures[i];

const int wh = f.Width + f.Height;

for (int y = 0; y + wh < sum.rows; y += wh) {

for (int x = f.Height; x + f.Width < sum.cols; x += wh) {

features.at<float>(0, featureNumber++) =

f.computeTilted(tilted, x, y);

}

}

}

if (featureNumber != count) {

std::cerr << "ERROR: featureNumber != count" << std::endl;

exit(1);

}

}

int EmoFeatures::featuresNumber(const cv::Size &imageSize) {

int count = 0;

for (size\_t i = 0; i < myFeatures.size(); ++i) {

const Feature &f = myFeatures[i];

int nx = imageSize.width / f.Width;

int ny = imageSize.height / f.Height;

count += nx \* ny;

int wh = f.Width + f.Height;

nx = imageSize.width / wh;

ny = imageSize.height / wh;

count += nx \* ny;

}

return count;

}

Объявление классов для чтения и записи данных метода «Анализа основных компонентов» в xml-файл:

class EmoPCAReader : public EmoXMLReader {

public:

static bool loadPCA(const std::string &fileName,

cv::Size &imageSize, cv::PCA &pca, float featuresArg);

private:

EmoPCAReader(cv::Size &imageSize, cv::PCA &pca, float featuresArg);

public:

void startElementHandler(const char \*tag, const char \*\*attributes);

void endElementHandler(const char \*tag);

void characterDataHandler(const char \*text, size\_t len);

private:

template <class \_MatType>

bool scanBuffer(cv::Mat &matrix, size\_t maxRows);

private:

cv::Size &myImageSize;

cv::PCA &myPca;

std::string myBuffer;

enum {

READ\_NOTHING,

READ\_PCA,

READ\_MEAN,

READ\_EIGENVALUES,

READ\_EIGENVECTORS,

READ\_SOURCE,

} myState;

size\_t myLoadCount;

float myLoadThreshold;

};

class EmoPCAWriter {

public:

EmoPCAWriter();

public:

bool write(const std::string &fileName,

const cv::Size &imageSize, const cv::PCA &pca);

private:

void writeSource(EmoXMLWriter &writer, const cv::Size &imageSize);

void writeMatrix(EmoXMLWriter &writer, const std::string &name,

const cv::Mat &matrix, int indent);

};

Для работы с xml-файлами были написаны классы EmoXMLReader и EmoXMLWriter, облегчающие чтение и запись информации на жесткий диск. Чтение xml-файлов реализовано при помощи открытой библиотеки expat (<http://expat.sourceforge.net/>).

# Приложение 2. Дополнительные средства

## Модуль emoextract

Используется для подготовки образцов входных воздействий. Позволяет на основе небольшого количества фотографий сгенерировать существенный объем изображений лиц. Процесс генерации совмещен с разметкой входных изображений информацией, необходимой для обучения.

Интерфейс модуля:

class ExtractProcessor {

public:

ExtractProcessor();

std::string init(const std::string &classesFile,

const std::string &configFile);

std::string processInput(const std::string &input);

private:

int readConfig(const std::map<std::string, std::string> &config);

static std::string baseName(const std::string &fileName);

bool processIndex(const std::string &input);

bool processImage(const cv::Mat &image, const std::string &base);

bool processFace(const cv::Mat &image, const cv::Rect &rect,

const std::string &base, int number);

char readClass();

int writeImage(const cv::Mat &image, const cv::Rect &rect,

const std::string &base, int number, int morphnumber, char letter);

int doScale(const cv::Mat &image, const cv::Rect &rect,

const std::string &base, int number, int morphnumber, char letter);

int doOffset(const cv::Mat &image, const cv::Rect &rect,

const std::string &base, int number, int morphnumber, char letter);

private:

int myMorphOffset;

int myMorphOffsetStep;

int myMorphScale;

int myMorphScaleStep;

float myStartScale;

std::vector<EmoClass> myClasses;

cv::CascadeClassifier myCvCascade;

float myScale;

float myMinFace;

std::string myOutFolder;

cv::Size myOutSize;

};

Реализация:

static const std::string imageWindow = "imageWindow";

std::string ExtractProcessor::processInput(const std::string &input) {

cvNamedWindow(imageWindow.c\_str(), 1);

if (!processIndex(input)) {

std::cerr << "ERROR: unable to process index file" << std::endl;

}

cvDestroyWindow(imageWindow.c\_str());

return "";

}

std::string ExtractProcessor::baseName(const std::string &fileName) {

size\_t index = fileName.rfind('/');

std::string name;

if (index == std::string::npos) {

name = fileName;

} else {

name = fileName.substr(index + 1);

}

index = name.rfind('.');

if (index == std::string::npos || index + 4 != name.length()) {

return name;

}

return name.substr(0, index);

}

bool ExtractProcessor::processIndex(const std::string &input) {

FILE\* f = fopen(input.c\_str(), "rt");

if (f == 0) {

return false;

}

std::cout << "Processing index file \"" << input << "\"" << std::endl;

cv::Mat frame;

char buf[1000];

while (fgets(buf, 1000, f)) {

int len = (int) strlen(buf);

while (len > 0 && isspace(buf[len-1])) {

len--;

}

buf[len] = '\0';

std::string imageName(buf);

std::string base = baseName(imageName);

std::cout << "file " << imageName.c\_str() << std::endl;

frame = cv::imread(imageName, 1);

if (!frame.empty()) {

if (!processImage(frame, base)) {

break;

}

} else {

std::cout << "WARNING: unable to read file" << std::endl;

std::cout << std::endl;

}

}

fclose(f);

return true;

}

bool ExtractProcessor::processImage(const cv::Mat &image,

const std::string &base) {

cv::Mat buffer;

cv::cvtColor(image, buffer, CV\_BGR2GRAY);

cv::equalizeHist(buffer, buffer);

std::vector<cv::Rect> faces;

myCvCascade.detectMultiScale(buffer, faces, myScale,

2, CV\_HAAR\_SCALE\_IMAGE,

cv::Size(buffer.cols \* myMinFace, buffer.rows \* myMinFace));

if (faces.empty()) {

cv::Mat canvas;

const double factor = 1.0 / myStartScale;

cv::resize(image, canvas, cv::Size(), factor, factor,

cv::INTER\_LINEAR);

cv::imshow(imageWindow, canvas);

std::cerr << "Unable to find faces" << std::endl;

std::cerr << std::endl;

cv::waitKey();

return true;

}

for (size\_t i = 0; i < faces.size(); ++i) {

if (!processFace(image, faces[i], base, i)) {

return false;

}

}

return true;

}

char ExtractProcessor::readClass() {

std::cout << "classes: " << std::endl;

for (size\_t i = 0; i < myClasses.size(); ++i) {

const EmoClass &cls = myClasses[i];

std::cout << "\tclass \"" << cls.name() << "\" -- " <<

cls.letter() << std::endl;

}

std::cout << "Press <ESC> to skip image" << std::endl;

while (true) {

std::cout << "Enter class: " << std::flush;

char letter = cv::waitKey();

if (letter == 27) {

std::cout << "<ESC>" << std::endl;

return '\0';

}

std::cout << letter << std::endl;

for (size\_t i = 0; i < myClasses.size(); ++i) {

const EmoClass &cls = myClasses[i];

if (letter == cls.letter()) {

return letter;

}

}

std::cout << "Unknown class" << std::endl;

}

}

bool ExtractProcessor::processFace(const cv::Mat &image,

const cv::Rect &rect, const std::string &base, int number) {

cv::Mat canvas;

const double factor = 1.0 / myStartScale;

cv::resize(image, canvas, cv::Size(), factor, factor, cv::INTER\_LINEAR);

cv::rectangle(

canvas,

cvPoint(rect.x \* factor, rect.y \* factor),

cvPoint((rect.x + rect.width) \* factor,

(rect.y + rect.height) \* factor),

CV\_RGB(255, 0, 255), 3);

cv::imshow(imageWindow, canvas);

while (true) {

std::cout << "action (q -- quit; s -- skip; z -- save): " <<

std::flush;

char ch = cv::waitKey(0);

std::cout << ch << std::endl;

if (ch == 'q') {

return false;

} else if (ch == 's') {

break;

} else if (ch != 'z') {

std::cout << "Unknown action: " << ch << std::endl;

continue;

}

std::cout << std::endl;

char letter = readClass();

if (letter == '\0') {

break;

}

std::cout << std::endl;

doScale(image, rect, base, number, 0, letter);

break;

}

return true;

}

int ExtractProcessor::writeImage(const cv::Mat &image, const cv::Rect &rect, const std::string &base, int number, int morphnumber, char letter) {

if (rect.x < 0 || rect.y < 0 || (rect.x + rect.width) > image.cols

|| (rect.y + rect.height) > image.rows) {

std::cerr << "\t\tout of range..." << std::endl;

return morphnumber;

}

cv::Mat face = image(rect);

std::ostringstream fns;

fns << myOutFolder << base << "\_";

fns << std::setfill('0') << std::setw(2) << number <<

std::setfill(' ') << std::setw(0);

fns << "\_";

fns << std::setfill('0') << std::setw(4) << morphnumber <<

std::setfill(' ') << std::setw(0);

fns << "-" << letter << ".ppm";

std::string fileName = fns.str();

cv::Mat buffer;

cv::resize(face, buffer, myOutSize, 0.f, 0.f, cv::INTER\_LINEAR);

cv::imwrite(fileName, buffer);

return morphnumber + 1;

}

int ExtractProcessor::doScale(const cv::Mat &image, const cv::Rect &rect,

const std::string &base, int number, int morphnumber, char letter) {

std::cerr << "SCALE: " << 0 << std::endl;

morphnumber = doOffset(image, rect, base, number, morphnumber, letter);

int scale = myMorphScaleStep;

while (scale <= myMorphScale) {

std::cerr << "SCALE: " << scale << std::endl;

cv::Rect scaled;

int dx = rect.width \* scale / 100;

int dy = rect.height \* scale / 100;

scaled = rect;

scaled.x -= dx / 2;

scaled.y -= dy / 2;

scaled.width += dx;

scaled.height += dy;

morphnumber = doOffset(image, scaled, base, number,

morphnumber, letter);

scaled = rect;

scaled.x += dx / 2;

scaled.y -= dy / 2;

scaled.width -= dx;

scaled.height += dy;

morphnumber = doOffset(image, scaled, base, number,

morphnumber, letter);

scaled = rect;

scaled.x -= dx / 2;

scaled.y += dy / 2;

scaled.width += dx;

scaled.height -= dy;

morphnumber = doOffset(image, scaled, base, number,

morphnumber, letter);

scaled = rect;

scaled.x += dx / 2;

scaled.y += dy / 2;

scaled.width -= dx;

scaled.height -= dy;

morphnumber = doOffset(image, scaled, base, number,

morphnumber, letter);

scaled = rect;

scaled.x -= dx / 2;

scaled.width += dx;

morphnumber = doOffset(image, scaled, base, number,

morphnumber, letter);

scaled = rect;

scaled.y -= dy / 2;

scaled.height += dy;

morphnumber = doOffset(image, scaled, base, number,

morphnumber, letter);

scaled = rect;

scaled.x += dx / 2;

scaled.width -= dx;

morphnumber = doOffset(image, scaled, base, number,

morphnumber, letter);

scaled = rect;

scaled.y += dy / 2;

scaled.height -= dy;

morphnumber = doOffset(image, scaled, base, number,

morphnumber, letter);

scale += myMorphScaleStep;

}

return morphnumber;

}

int ExtractProcessor::doOffset(const cv::Mat &image, const cv::Rect &rect, const std::string &base, int number, int morphnumber, char letter) {

std::cerr << "\tOFFSET: " << 0 << std::endl;

morphnumber = writeImage(image, rect, base, number,

morphnumber, letter);

int offset = myMorphOffsetStep;

while (offset <= myMorphOffset) {

std::cerr << "\tOFFSET: " << offset << std::endl;

int dx = rect.width \* offset / 100;

int dy = rect.height \* offset / 100;

cv::Rect face;

face = rect;

face.x += dx;

face.y += dy;

morphnumber = writeImage(image, face, base, number,

morphnumber, letter);

face = rect;

face.x -= dx;

face.y += dy;

morphnumber = writeImage(image, face, base, number,

morphnumber, letter);

face = rect;

face.x += dx;

face.y -= dy;

morphnumber = writeImage(image, face, base, number,

morphnumber, letter);

face = rect;

face.x -= dx;

face.y -= dy;

morphnumber = writeImage(image, face, base, number,

morphnumber, letter);

face = rect;

face.x += dx;

morphnumber = writeImage(image, face, base, number,

morphnumber, letter);

face = rect;

face.y += dy;

morphnumber = writeImage(image, face, base, number,

morphnumber, letter);

face = rect;

face.x -= dx;

morphnumber = writeImage(image, face, base, number,

morphnumber, letter);

face = rect;

face.y -= dy;

morphnumber = writeImage(image, face, base, number,

morphnumber, letter);

offset += myMorphOffsetStep;

}

return morphnumber;

}

## Модуль emoeigen

Используется для подготовки данных для метода «Анализа основных компонентов». В этом модуле реализовано выделение основных компонентов из множества входных изображений с сохранением в xml-файл.

Интерфейс модуля:

class EigenProcessor {

public:

EigenProcessor();

std::string init(const std::string &config);

std::string processInput(const std::string &input);

std::string saveToFile(const std::string &pca);

private:

int readConfig(const std::map<std::string, std::string> &config);

bool fillSamples(const std::string &input, cv::Mat &samples);

private:

int myFaceWidth;

int myFaceHeight;

int myMaxComponents;

cv::PCA myCvPCA;

};

Реализация:

std::string EigenProcessor::processInput(const std::string &input) {

cv::Mat samples;

if (!fillSamples(input, samples)) {

return "Unable to read samples";

}

myCvPCA(samples, cv::Mat(), CV\_PCA\_DATA\_AS\_ROW, myMaxComponents);

return "";

}

bool EigenProcessor::fillSamples(const std::string &input, cv::Mat &samples) {

FILE\* f = fopen(input.c\_str(), "rt");

if (f == 0) {

return false;

}

cv::Mat image, gray, resized, resizedFloat;

std::vector<cv::Mat> imagesVector;

char buf[1000];

while (fgets(buf, 1000, f)) {

int len = (int) strlen(buf);

while (len > 0 && isspace(buf[len-1])) {

len--;

}

buf[len] = '\0';

std::string imageName(buf);

std::cout << "file " << imageName.c\_str() << std::endl;

image = cv::imread(imageName, 1);

if (!image.empty()) {

cv::cvtColor(image, gray, CV\_BGR2GRAY);

cv::resize(gray, resized,

cv::Size(myFaceWidth, myFaceHeight),

0, 0, cv::INTER\_LINEAR);

cv::equalizeHist(resized, resized);

resized.convertTo(resizedFloat, CV\_32FC1, 1.0 / 255);

imagesVector.push\_back(cv::Mat());

cv::Mat &vec = imagesVector.back();

resizedFloat.reshape(0, 1).copyTo(vec);

} else {

std::cout << "WARNING: unable to read file" << std::endl;

std::cout << std::endl;

}

}

fclose(f);

samples.create(imagesVector.size(), myFaceWidth\*myFaceHeight, CV\_32FC1);

for (size\_t i = 0; i < imagesVector.size(); ++i) {

cv::Mat rowi = samples.row(i);

imagesVector[i].copyTo(rowi);

}

return true;

}

std::string EigenProcessor::saveToFile(const std::string &pca) {

EmoPCAWriter writer;

std::cout << "Saving PCA to file: " << pca << std::endl;

if (!writer.write(pca, cv::Size(myFaceWidth, myFaceHeight), myCvPCA)) {

return "Unable to write PCA data to file";

}

return "";

}

## Модуль emoshow

Используется для визуального отображения данных метода «Анализа основных компонентов». В этом модуле реализована загрузка выделенных основных компонент из файла и преобразование в форму, доступную для визуального отображения.

Интерфейс модуля:

class ShowProcessor {

public:

ShowProcessor();

std::string init(const std::string &pcaFile);

std::string exec();

void showFace(int pos);

private:

cv::PCA myCvPCA;

cv::Size myFaceSize;

};

Реализация:

std::string ShowProcessor::init(const std::string &pcaFile) {

if (!EmoPCAReader::loadPCA(pcaFile, myFaceSize, myCvPCA, -1.0f)) {

return "Unable to load PCA";

}

return "";

}

static const std::string imageWindow = "imageWindow";

static const std::string positionBar = "positionBar";

static void positionCallBack(int pos, void \*data) {

((ShowProcessor\*)data)->showFace(pos);

}

std::string ShowProcessor::exec() {

cvNamedWindow(imageWindow.c\_str(), 1);

cv::createTrackbar(positionBar, imageWindow, 0,

myCvPCA.eigenvectors.rows - 1, &positionCallBack, this);

std::cout << "Press <ESC> to quit..." << std::endl;

while (cv::waitKey() != 27);

cvDestroyWindow(imageWindow.c\_str());

return "";

}

void ShowProcessor::showFace(int pos) {

if (pos < 0 || pos >= myCvPCA.eigenvectors.rows) {

std::cerr << "WARNING: invalid slider position: " << pos

<< std::endl;

return;

}

cv::Mat buffer;

myCvPCA.eigenvectors.row(pos).reshape(0, myFaceSize.height)

.copyTo(buffer);

cv::normalize(buffer, buffer, 0.0, 1.0, cv::NORM\_MINMAX);

cv::imshow(imageWindow, buffer);

}

## Модуль emotrain

Используется для тренировки модели многослойного персептрона на основе набора изображений и данных метода преобразования изображений в вектор признаков.

Интерфейс модуля:

class TrainProcessor {

public:

TrainProcessor();

std::string init(const std::string &configFile,

const std::string &classesFile, const std::string &pcaFile);

std::string processInput(const std::string &input);

std::string saveToFile(const std::string &mlp);

private:

int readConfig(const std::map<std::string, std::string> &config);

static char category(const std::string &fileName);

bool fillSamples(const std::string &inputName, cv::Mat &samples,

cv::Mat &categories);

private:

std::string myClassesList;

float myFeatures;

cv::PCA myCvPCA;

cv::Size myFaceSize;

float myLayersScale;

int myLayersCount;

float myLayersSizeScale;

CvANN\_MLP myCvMLP;

};

Реализация (для метода «Анализа основных компонент»):

static const std::string imageWindow = "imageWindow";

std::string TrainProcessor::processInput(const std::string &input) {

cv::Mat samples, categories;

if (!fillSamples(input, samples, categories)) {

return "Unable to load samples from file";

}

const int inputSize = myCvPCA.eigenvalues.rows;

const int outputSize = myClassesList.length();

cv::Mat inputs;

myCvPCA.project(samples, inputs);

cv::Mat outputs(samples.rows, outputSize, CV\_32FC1);

outputs = 0.0f;

for (int i = 0; i < categories.rows; ++i) {

char cat = categories.at<unsigned char>(i, 0);

int index = (int) myClassesList.find(cat);

outputs.at<float>(i, index) = 1.0f;

}

int layers = (myLayersCount > 0) ? myLayersCount :

std::max(3, (int)(inputSize \* myLayersScale));

std::cout << std::endl;

std::cout << "Layers number = " << layers << std::endl;

cv::Mat layerSizes(1, layers, CV\_32SC1);

--layers;

std::cout << "Layer sizes: " << inputSize;

layerSizes.at<int>(0, 0) = inputSize;

for (int i = 1; i < layers; ++i) {

const float scale = myLayersSizeScale +

(1.0f - myLayersSizeScale) \* (i-1) / (layers-1);

const int sz = (int)(scale \* (inputSize +

(outputSize - inputSize) \* i / layers));

std::cout << " " << sz;

layerSizes.at<int>(0, i) = sz;

}

std::cout << " " << outputSize << std::endl;

layerSizes.at<int>(0, layers) = outputSize;

std::cout << std::endl;

double timer = (double)cv::getTickCount();

myCvMLP.create(layerSizes, CvANN\_MLP::SIGMOID\_SYM, 1.0, 1.0);

myCvMLP.train(inputs, outputs, cv::Mat(), cv::Mat(),

CvANN\_MLP\_TrainParams(), 0);

timer = (double)cv::getTickCount() - timer;

std::cout << "Training time = " << (timer / cv::getTickFrequency()) <<

" s" << std::endl;

std::cout << std::endl;

return "";

}

bool TrainProcessor::fillSamples(const std::string &inputName,

cv::Mat &samples, cv::Mat &categories) {

FILE\* f = fopen(inputName.c\_str(), "rt");

if (f == 0) {

return false;

}

cv::Mat image, gray, resized, resizedFloat;

std::vector<cv::Mat> imagesVector;

std::vector<char> categoriesVector;

char buf[1000];

while (fgets(buf, 1000, f)) {

int len = (int) strlen(buf);

while (len > 0 && isspace(buf[len-1])) {

len--;

}

buf[len] = '\0';

std::string imageName(buf);

char cat = category(imageName);

std::cout << "file " << imageName.c\_str() << std::endl;

if (cat == '\0') {

std::cout << "WARNING: no category detected" << std::endl;

std::cout << std::endl;

continue;

} else if (myClassesList.find(cat) == std::string::npos) {

std::cout << "WARNING: unknown category detected" <<

std::endl;

std::cout << std::endl;

continue;

}

image = cv::imread(imageName, 1);

if (!image.empty()) {

cv::cvtColor(image, gray, CV\_BGR2GRAY);

cv::resize(gray, resized, myFaceSize, 0, 0,

cv::INTER\_LINEAR);

cv::equalizeHist(resized, resized);

resized.convertTo(resizedFloat, CV\_32FC1, 1.0 / 255);

imagesVector.push\_back(cv::Mat());

cv::Mat &vec = imagesVector.back();

resizedFloat.reshape(0, 1).copyTo(vec);

categoriesVector.push\_back(cat);

} else {

std::cout << "WARNING: unable to read file" << std::endl;

std::cout << std::endl;

}

}

fclose(f);

samples.create(imagesVector.size(),

myFaceSize.width \* myFaceSize.height, CV\_32FC1);

categories.create(imagesVector.size(), 1, CV\_8UC1);

for (size\_t i = 0; i < imagesVector.size(); ++i) {

cv::Mat rowi = samples.row(i);

imagesVector[i].copyTo(rowi);

categories.at<unsigned char>(i, 0) = categoriesVector.at(i);

}

return true;

}

char TrainProcessor::category(const std::string &fileName) {

size\_t line = fileName.rfind('-');

if (line == std::string::npos) {

return '\0';

}

line += 2;

if (line == fileName.length() || line == fileName.find('.', line)) {

return fileName.at(line - 1);

}

return '\0';

}

std::string TrainProcessor::saveToFile(const std::string &mlp) {

std::cout << "Saving MLP to file: " << mlp << std::endl;

myCvMLP.save(mlp.c\_str());

return "";

}