Нижегородский государственный университет им. Н.И. Лобачевского
Факультет вычислительной математики и кибернетики

Учебный курс «Разработка мультимедийных приложений с использованием библиотек OpenCV и IPP»

Лабораторная работа Классификация изображений с использованием bag-of-words методов

Половинкин А.Н.

При поддержке компании Intel

Нижний Новгород
2012

Содержание

ВВЕДЕНИЕ **4**

1. МЕТОДИЧЕСКИЕ УКАЗАНИЯ **5**

1.1. Цели и задачи работы 5
1.2. Структура работы 5
1.3. Тестовая инфраструктура 6
1.4. Требования к участнику лабораторной работы 6
1.5. Рекомендации по проведению занятий 6

2. ОБЗОР ВОЗМОЖНОСТЕЙ МОДУЛЯ FEATURES2D БИБЛИОТЕКИ OPENCV **7**

2.1. Детектирование ключевых точек на изображении 7
Для того, чтобы найти на изображении ключевые точки с использованием созданного детектора, необходимо вызвать метод `detect` 8
2.2. Вычисление детекторов ключевых точек на изображении 9
2.3. Обучение словаря с использованием алгоритма кластеризации K-means 12
2.4. Вычисление признаковых описаний изображений 13

3. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ **16**

3.1. Разработка программы классификации изображений с использованием bag-of-words методов 16

4. КОНТРОЛЬНЫЕ ВОПРОСЫ **24**

5. ДОПОЛНИТЕЛЬНЫЕ ЗАДАНИЯ **25**

6. ЛИТЕРАТУРА **25**

6.1. Основная литература 25
6.2. Ресурсы сети Интернет 26

7. ПРИЛОЖЕНИЯ **27**

7.1. Приложение А. Исходный код основной функции консольного редактора изображений 27
7.2. Приложение Б. Исходный код заголовочного файла модуля, содержащего вспомогательные функции 27
7.3. Приложение В. Исходный код модуля, содержащего вспомогательные функции 27
7.4. Приложение Г. Исходный код заголовочного файла модуля, содержащего реализацию основных стадий bag-of-words подхода 28

7.5. Приложение Д. Исходный код модуля, содержащего реализацию основных стадий bag-of-words подхода 29

Введение

В лабораторной работе рассматривается один из наиболее распространенных классов алгоритмов классификации изображений: так называемые bag-of-words (также известные как bag-of-features или bag-of-keypoints) методы [13]. Фактически bag-of-words подход использует в качестве описания гистограмму вхождений отдельных шаблонов в изображение. Корни идеи восходят к задаче классификации текстов, для решения которой используют описания в виде гистограмм вхождений в документ слов из заранее составленного словаря.

Основные шаги алгоритмов класса bag-of-words могут быть описаны следующим образом:

1. Детектирование ключевых точек на изображении.
2. Построение описания (дескрипторов) локальных окрестностей ключевых точек.
3. Кластеризация дескрипторов ключевых точек, принадлежащих всем объектам обучающей выборки (это соответствует построению словаря, «словами» в котором являются центроиды построенных кластеров).
4. Построение описания каждого изображения в виде нормированной гистограммы встречаемости «слов» (для каждого кластера вычисляется количество отнесенных к нему ключевых точек, принадлежащих определенному изображению).
5. Построение классификатора, использующего вычисленное на шаге 4 признаковое описание изображения.

На дескрипторы, используемые алгоритмами данного класса, накладываются определенные ограничения: в частности, они должны быть инварианты к аффинным преобразованиям изображения, изменениям в условиях освещенности и перекрытиям. Словарь дескрипторов ключевых точек должен быть достаточно большим, чтобы отражать релевантные изменения частей изображения, но в то же время не чрезмерным, чтобы сделать алгоритм устойчивым к шуму.

В настоящей работе рассматривается ряд методов детектирования ключевых точек и построения их дескрипторов, обучения словаря с использованием алгоритма кластеризации k-средних(KMeans) [5], а также построения признаковых описаний объектов на основе данного словаря. Приводятся примеры использования алгоритмов обучения с учителем для классификации изображений с использованием вычисленных признаковых описаний. В работе демонстрируется использование соответствующих операций на базе средств открытой библиотеки компьютерного зрения OpenCV [19].

1. Методические указания

1.1. Цели и задачи работы

Цель данной работы – изучить bag-of-words подход для классификации изображений с использованием соответствующих функций библиотеки компьютерного зрения OpenCV.

Данная цель предполагает решение следующих задач:

1. Изучить bag-of-words подход для классификации изображений и принцип работы базовых операций обработки изображений:
   - детектирование ключевых точек на изображении;
   - вычисление дескрипторов ключевых точек;
   - построение словаря дескрипторов ключевых точек;
   - вычисление признакового описания изображений с использованием словаря дескрипторов ключевых точек;
   - обучение классификатора «случайный лес» и предсказание категорий новых изображений.
2. Рассмотреть прототипы функций, реализующих перечисленные операции в библиотеке OpenCV.
3. Разработать простые примеры использования указанного набора функций.
4. Разработать консольное приложение, содержащее реализацию bag-of-words подхода для задачи классификации изображений двух категорий.
5. Провести вычислительный эксперимент на основе подмножества изображений из набора данных Caltech-101 [14].
6. Выполнить исследование зависимости ошибки классификации от используемых параметров (типа используемых детекторов и дескрипторов ключевых точек, числа слов в словаре, параметров алгоритма обучения с учителем «случайный лес»).
7. Реализовать подход для построения словаря на основе Gaussian Mixture Model [5] и сравнить полученные результаты с ранее реализованным подходом.

1.2. Структура работы

В работе предлагается описание основных шагов bag-of-words подхода для классификации изображений. Рассматриваются прототипы функций

библиотеки OpenCV, необходимые для реализации данных шагов, с описанием назначения входных параметров. Предлагаются примеры подпрограмм, демонстрирующие использование каждой функции. Разрабатывается приложение, содержащее реализацию bag-of-words подхода для классификации двух категорий изображений. Проводится вычислительный эксперимент на наборе данных, применяемом на практике. Рассматривается зависимость результатов применения подхода от используемых параметров.

1.3. Тестовая инфраструктура

Вычислительные эксперименты проводились с использованием следующей инфраструктуры (табл. 1).

Таблица 1. Тестовая инфраструктура

Операционная система	Microsoft Windows 7
Среда разработки	Microsoft Visual Studio 2010
Библиотеки OpenCV	Версия 2.4.3

1.4. Требования к участнику лабораторной работы

Необходимо предварительное изучение участником работы теоретического материала по следующим темам:

- классификация изображений;
- детекторы и дескрипторы ключевых точек;
- алгоритмы обучения с учителем.

Также требуется ознакомление с материалами лабораторной работы «Машинное обучение в библиотеке OpenCV».

1.5. Рекомендации по проведению занятий

При выполнении данной лабораторной работы рекомендуется следующая последовательность действий:

1. Привести примеры задач и приложений компьютерного зрения, которые требуют решения задачи классификации изображений.
2. Последовательно рассмотреть основные шаги bag-of-words подхода для классификации изображений. Параллельно ввести функции библиотеки OpenCV, необходимые для реализации данных шагов, и продемонстрировать примеры подпрограмм с их применением.

3. Рассмотреть структуру приложения, реализующего bag-of-words подход для классификации изображений. Объяснить назначение всех функций, которые небходимо реализовать.
4. Рассмотреть параметры запуска приложения, необходимые для получения результатов вычислительного эксперимента.
5. Рассмотреть задание, связанное с исследованием результатов классификации объектов от параметров алгоритма.
6. Рассмотреть подход для построения словаря на основе Gaussian Mixture Model. Сформулировать задание, связанное с реализацией данного подхода в рамках лабораторной работы.

2. Обзор возможностей модуля features2d библиотеки OpenCV

2.1. Детектирование ключевых точек на изображении

Ключевое значение при построении признакового описания изображения имеет выделение (детектирование) характерных для изображения частей, в качестве которых можно рассмотреть, например, углы, ребра, регионы, соответствующие экстремумам интенсивности, и т.п. На алгоритмы, выделяющие такие особенности (ключевые точки), накладываются требования инвариантности относительно преобразований смещения и поворота, изменения масштаба и освещенности исходного изображения, а также точки положения камеры, с которой снят объект.

Одним из наиболее простых алгоритмов, который находит на изображении углы, является *детектор Моравеца* (Moravec) [15]. Основная идея алгоритма заключается в поиске точек, соответствующих максимуму изменения суммарной интенсивности в окрестности рассматриваемого пикселя при смещении данной окрестности на один пиксель в каждом из восьми принципиальных направлений (2 горизонтальных, 2 вертикальных и 4 диагональных). Основными недостатками данного детектора являются отсутствие инвариантности к преобразованию типа «поворот» и возникновение большого числа ложных срабатываний на диагональных ребрах. Детектор Моравеца обладает свойством анизотропии в 8 принципиальных направлениях смещения окна.

Детектор Харриса (Harris) [16] строится на основании детектора Моравеца и является его улучшением. По сравнению со своим предшественником он инвариантен относительно поворота, количество ложных срабатываний не велико за счет введения свертки с Гауссовыми весовыми коэффициентами. Однако результаты поиска значительно меняются при масштабировании изображения. Также существуют

модификации детектора Харриса, которые учитывают вторые производные функции интенсивности (например, детектор Харриса-Лапласа (Harris-Laplace) [10]).

Детектор MSER (Maximally Stable Extremal Regions) [9] выделяет множество регионов, являющихся областями локального экстремума интенсивности, максимально устойчивыми к вариациям интенсивности в окрестности данных регионов. Детектор обладает свойством инвариантности относительно масштабирования изображения. Похожая идея поиска областей с экстремальными свойствами интенсивности пикселей, устойчивых к изменению масштаба, лежит в основе *детекторов LoG* (Laplacian of Gaussian [7]) и *SIFT* [8]).

Приведенные выше детекторы определяют расположение ключевых точек на изображении, используя только пиксели исходного изображения. Альтернативный подход состоит в том, чтобы использовать алгоритмы машинного обучения для тренировки классификатора точек на некотором множестве изображений. *FAST-детектор* (Features from Accelerated Test) [11] является типичным представителем данного класса детекторов.

Более подробное описание детекторов ключевых точек приведено в лекционной части курса.

В библиотеке OpenCV реализованы следующие детекторы: FAST [11], STAR [1], SIFT [8], SURF [2], ORB [12], MSER [9], GoodFeaturesToTrack [20], Harris [16], Dense [17], Simple blob [17]. Для использования детекторов особых точек необходимо подключить заголовочный файл opencv2/features2d/features2d.hpp и библиотеку features2d243(d).lib.

Классы, содержащие реализацию данных детекторов, унаследованы от базового абстрактного класса **`FeatureDetector`**. Рассмотрим подробнее методы данного класса.

Для создания детектора ключевых точек заданного типа используется метод **`create`**, возвращающий указатель на созданный объект класса детектора.

```
Ptr<FeatureDetector> FeatureDetector::create(
                                  const string& detectorType)
```

В качестве параметра данной функции передается строка, содержащая идентификатор типа детектора (“FAST”, “STAR”, “SIFT”, “SURF”, “ORB”, “MSER”, “GFTT” (GoodFeaturesToTrack), “HARRIS”, “Dense”, “SimpleBlob”.

Для того, чтобы найти на изображении ключевые точки с использованием созданного детектора, необходимо вызвать метод `detect`

```
void FeatureDetector::detect(const Mat& image,
```

```
                        vector<KeyPoint>& keypoints,
                        const Mat& mask=Mat() ) const
```

Рассмотрим параметры данного метода:

- **image** – входное изображение
- **keypoints** – массив найденных особых точек на изображении.
- **mask** – маска, определяющая область изображения, в которой осуществляется поиск ключевых точек (элементами матрицы, описывающей маску, являются восьмибитные целые числа; ненулевые значения соответствуют области изображения).

Необходимо учесть, что при использовании детекторов SIFT и SURF требуется в исходном коде дополнительно подключить заголовочный файл opencv2/nonfree/nonfree.hpp, а в настройках проекта библиотеку nonfree243(d).lib. Наряду с этим вызвать функцию инициализации данного модуля: initModule_nonfree().

Рассмотрим пример функции, которая принимает на вход путь к изображению и возвращает найденные на данном изображении ключевые точки с использованием детектора SIFT.

```
// fileName – путь к изображению
// keypoints – найденные ключевые точки на изображении
void DetectKeypointsOnImage(const string& fileName,
                            vector<KeyPoint>& keypoints)
{
      // загружаем изображение из файла
      Mat img = imread(fileName);
      // инициализируем модуль nonfree для использования
      // детектора SIFT
      initModule_nonfree();
      // создаем SIFT детектор
      Ptr<FeatureDetector> featureDetector =
                        FeatureDetector::create("SIFT");
      // детектируем ключевые точки
      // на загруженном изображении
      featureDetector->detect(img, keypoints);
}
```

2.2. Вычисление детекторов ключевых точек на изображении

Дескриптором ключевой точки называется числовой вектор, характеризующий особенности изображения в окрестности данной точки. Изначально процедура построения дескрипторов применялась для нахождения соотвествия между ключевыми точками на разных изображениях. В результате построения формируется множество векторов признаков для исходного набора особых точек.

Одним из наиболее известных дескрипторов ключевых точек является *SIFT* (Scale Invariant Feature Transform) [8], в основе которого лежит идея вычисления гистограммы ориентированных градиентов в окрестности особой точки. Модификацией данного дескриптора является PCA-SIFT [6], который строится по той же схеме, используя окрестность большего размера. Для результирующего набора дескрипторов осуществляется снижение размерности векторов посредством анализа главных компонент (Principal Component Analysis, PCA) [5].

Дескриптор SURF (Speeded up Robust Features) [2] также относится к числу тех дескрипторов, которые одновременно выполняют поиск особых точек и строят их описание, инвариантное к изменению масштаба и повороту изображения. Фактически данный дескриптор представляет собой взвешенные значения градиентов и их абсолютных значений в 16 квадрантах, на которые разбита окрестность особой точки.

Цель создания *дескриптора BRIEF* (Binary Robust Independent Elementary Features) [4] состояла в том, чтобы обеспечить распознавание одинаковых участков изображения, которые были получены с разных углов обзора. При этом ставилась задача уменьшить количество выполняемых вычислений. В окрестности точки выбирается некоторым образом множество пар пикселей, и на данном множестве строится набор бинарных тестов, представляющих собой сравнение интенсивности в соответствующих пикселях. Результаты тестов объединяются в битовую строку, которая и является итоговым дескриптором ключевой точки. Более эффективной альтернативой дескриптору BRIEF является бинарный *дескриптор ORB* [12].

Подробное описание дескрипторов ключевых точек можно найти в лекционной части курса.

В библиотеке OpenCV реализовано вычисление следующих дескрипторов ключевых точек: SIFT, SURF, BRIEF, ORB, для использования которых необходимо в исходном коде приложения подключить заголовочный файл opencv2/features2d/features2d.hpp, а в настройках проекта библиотеку features2d243(d).lib

Классы, содержащие реализацию алгоритмов вычисления данных дескрипторов, унаследованы от базового абстрактного класса **`DescriptorExtractor`**. Рассмотрим подробнее методы данного класса.

Для создания объекта класса, предназначенного для вычисления дескрипторов ключевых точек заданного типа, используется метод **`create`**, возвращающий указатель на созданный объект класса

```
Ptr<DescriptorExtractor> DescriptorExtractor::create(
                    const string& descriptorExtractorType)
```

В качестве параметра данной функции передается строка, содержащая идентификатор типа дескриптора (“SIFT”, “SURF”, “BRIEF”, “ORB”).

Для того чтобы вычислить дескрипторы ключевых точек на изображении с использованием созданного детектора, необходимо вызвать метод **compute**.

```
void    DescriptorExtractor::compute(const    Mat&    image,
                                vector<KeyPoint>& keypoints,
                                Mat& descriptors) const
```

Рассмотрим параметры данного метода:

- **image** – входное изображение.
- **keypoints** – массив ключевых точек на изображении.
- **descriptors** – вычисленные значения дескрипторов ключевых точек.

Необходимо учесть, что при использовании дескрипторов SIFT и SURF требуется дополнительно подключить заголовочный файл opencv2/nonfree/nonfree.hpp и библиотеку opencv_nonfree243(d).lib, а также вызвать функцию инициализации данного модуля: initModule_nonfree().

Рассмотрим пример функции, которая принимает на вход изображение и заранее найденные на нем ключевые точки и возвращает вычисленные значения SIFT дескрипторов в данных точках.

```
// img – исходное изображение
// keypoints – ключевые точки на изображении
// descriptors – вычисленные значения дескрипторов
//               ключевых точек
void ComputeKeypointDescriptorsOnImage(
                  const string& fileName,
                  vector<KeyPoint>& keypoints,
                  Mat& descriptors)
{
      // инициализируем модуль nonfree для использования
      // дескрипторов SIFT
      // (если данная функция ранее не вызывалась)
      initModule_nonfree();
      // создаем объект класса вычисления SIFT дескрипторов
      Ptr<DescriptorExtractor> descExtractor =
                    DescriptorExtractor::create(“SIFT”);
      // вычисляем дескрипторы ключевых точек
      // на загруженном изображении
      descExtractor->compute(img, keypoints, descriptors);
}
```

2.3. Обучение словаря с использованием алгоритма кластеризации K-means

Одним из основных подходов к обучению словаря дескрипторов является использование алгоритмов кластеризации, в частности, K-means [5], который разбивает множество объектов на заранее известное число кластеров. В основе данного алгоритма лежит итерационная процедура, в рамках которой вычисляются центроиды кластеров (как математическое ожидание относящихся к данным кластерам объектов), а потом обновляется информация о принадлежности каждого объекта к кластерам путем выбора ближайшего центроида.

Для обучения словаря с использованием алгоритма K-means в библиотеке OpenCV реализован класс **BOWKMeansTrainer,** унаследованный от базового абстрактного класса **BOWKMeansTrainer**. Рассмотрим методы данного класса.

```
BOWKMeansTrainer(int clusterCount,
      const TermCriteria&   termcrit = TermCriteria(),
      int attempts=3, int flags=KMEANS_PP_CENTERS)
```

Параметры конструктора данного класса соответствуют параметрам функции **kmeans** библиотеки OpenCV.

- **clusterCount** – число слов в словаре (соответствует числу кластеров в алгоритме K-means).
- **termcrit** – критерий остановки алгоритма кластеризации (максимальное число итераций или(и) достижение заданной точности). Точность задается в соответствующем поле **criteria.epsilon**. Если в результате очередной итерации алгоритма центроиды изменились на величину, меньше данного параметра, то алгоритм останавливает работу.
- **attempts** – число запусков алгоритма кластеризации со случайно выбранными центроидами на начальной итерации алгоритма.
- **flags** – параметр, определяющий способ инициализации центроидов на начальной итерации алгоритма. Возможные значения:
    - **KMEANS_RANDOM_CENTERS** – использовать случайно выбранные центроиды.
    - **KMEANS_PP_CENTERS** – использовать эвристику Arthur and Vassilvitskii [5]. Данный параметр показал лучшую эффективность на практике и используется по умолчанию.

Для формирования выборки объектов, на которой будет проходить обучение словаря, должен быть использован метод **add**:

```
void BOWTrainer::add(const Mat& descriptors)
```

В качестве параметра данной функции передается набор (матрица) дескрипторов особых точек, вычисленных ранее.

После того, как выборка сформирована, обучение словаря выполняется с использованием метода **cluster**.

```
Mat BOWTrainer::cluster()
```

Данный метод возвращает набор центроидов построенных кластеров, который в дальнейшем используется в качестве словаря дескрипторов особых точек.

Рассмотрим пример функции, которая принимает на вход набор вычисленных дескрипторов особых точек и размер словаря, и возвращает построенный словарь.

```
// descriptors – предвычисленный набор дескрипторов
// изображений, используемый при построении словаря;
// vocSize – размер словаря;
Mat  BuildVocabulary(const  std::vector<Mat>&  descriptors,
int vocSize)
{
      // создаем объект класса BOWTrainer с числом
      // кластеров, равным vocSize, и остальными
      // параметрами, используемыми по умолчанию
      BOWKMeansTrainer bowTrainer(vocSize);
      // добавляем дескрипторы особых точек с изображений,
      // используемых при обучении словаря
      for (size_t i = 0; i < descriptors.size(); i++)
      {
            bowTrainer.add(descriptors[i]);
      }
      // вызываем метод обучения словаря
     Mat voc = bowTrainer.cluster();
     return voc;
}
```

2.4. Вычисление признаковых описаний изображений

Большинство известных алгоритмов обучения с учителем требует описания каждого объекта в виде векторов одинаковой длины. В связи с этим напрямую использовать вычисленные значения дескрипторов, объединенные в один вектор, для решения задачи классификации не представляется возможным, т.к. в общем случае число ключевых точек на изображениях различно. К тому же, такое представление в любом случае не обладало бы инвариантностью относительно многих преобразований

изображения, т.к. при этом обычно изменяется расположение данных точек и, следовательно, расположение дескрипторов внутри объединенного вектора. Данная проблема решается при использовании bag-of-words подхода [13], в котором изображение описывается не самими дескрипторами, а частотами встречаемости их различных типов на изображении. Типы дескрипторов описываются словарем, который содержит центроиды кластеров дескрипторов особых точек. Таким образом, в случае, когда словарь уже вычислен, признаковое описание изображения вычисляется по следующей схеме:

- детектирование ключевых точек на изображении и вычисление их дескрипторов;
- нахождение для каждого вычисленного дескриптора ближайшего к нему центроида кластера;
- вычисление признакового описания изображения в виде нормированной гистограммы (i-ый бин гистограммы соответствует числу вхождений i-го слова из словаря в описание изображения, что соотвествует числу дескрипторов, отнесенных к i-му кластеру).

Библиотека OpenCV содержит класс **BOWImgDescriptorExtractor**, реализующий данную схему. Рассмотрим подробнее методы данного класса.

```
BOWImgDescriptorExtractor::BOWImgDescriptorExtractor(
            const Ptr<DescriptorExtractor>& dextractor,
            const Ptr<DescriptorMatcher>& dmatcher)
```

Рассмотрим параметры конструктора данного класса:

- **dextractor** – алгоритм, вычисляющий дескрипторы в ключевых точках
- **dmatcher** – алгоритм, используемый для нахождения ближайшего центроида к текущему дескриптору ключевой точки (обычно используется L2 метрика)

Для того чтобы задать словарь, используемый при вычислении признакового описания изображения, необходимо вызвать метод **setVocabulary**

```
void BOWImgDescriptorExtractor::setVocabulary(const Mat&
vocabulary)
```

В качестве параметра данному методу передается матрица, содержащая центроиды кластеров (обычно это матрица возвращается методом **cluster** класса **BOWTrainer**).

Вычисление признакового описания изображения осуществляется при помощи метода **compute**.

```
void BOWImgDescriptorExtractor::compute(const Mat& image,
            vector<KeyPoint>& keypoints,
```

```
            Mat& imgDescriptor,
            vector<vector<int>>* pointIdxsOfClusters=0,
            Mat* descriptors=0 )
```

Рассмотрим параметры данного метода:

- **image** – входное изображение.
- **keypoints** – массив ключевых точек на изображении, в которых вычисляются дескрипторы.
- **imgDescriptor** – вычисленное признаковое описание изображения.
- **pointIdxsOfClusters** – набор индексов дескрипторов, относящихся к тому или иному кластеру.
- **descriptors** – вычисленные значения дескрипторов ключевых точек.

Два последних параметра являются необязательными.

Рассмотрим пример функции, которая принимает на вход изображение и словарь дескрипторов ключевых точек и возвращает вычисленное признаковое описание изображения

```
// img – исходное изображение
// voc – словарь дескрипторов ключевых точек;
// imgDesc – вычисленное признаковое описание изображения
void ComputeImgDescriptor(Mat& img, Mat& voc, Mat& imgDesc)
{
      // создаем SIFT детектор ключевых точек
      Ptr<FeatureDetector> featureDetector =
                          FeatureDetector::create(“SIFT”);
      // создаем объект класса вычисления
      // SIFT дескрипторов ключевых точек
      Ptr<DescriptorExtractor> dExtractor =
                     DescriptorExtractor::create(“SIFT”);
      // создаем объект класса, находящего ближайший
      // к дескриптору центроид (по L2 метрике)
      Ptr<DescriptorMatcher> descriptorsMatcher =
                DescriptorMatcher::create("BruteForce");
      // создаем объект класса, вычисляющего
      // признаковое описание изображений
      Ptr<BOWImgDescriptorExtractor> bowExtractor = new
                               BOWImgDescriptorExtractor(
                                     dExtractor,
                                     descriptorsMatcher);
      // устанавливаем используемый словарь
      // дескрипторов ключевых точек
      bowExtractor->setVocabulary(voc);
      // находим ключевые точки на изображении
```

```
        vector<KeyPoint> keypoints;
        featureDetector->detect(img, keypoints);
        // вычисляем признаковое описание изображения
        bowExtractor->compute(img, keypoints, imgDesc);
}
```

3. Программная реализация

3.1. Разработка программы классификации изображений с использованием bag-of-words методов

3.1.1. Требования к приложению

На данном этапе предлагается разработать приложение, предназначенное для классификации изображений двух различных классов, предусматривающее возможность применения всех операций, перечисленных в разделе 2. К приложению предъявляются следующие требования:

1. Ввод исходных данных (пути к папкам, содержащим изображения обоих классов; тип детектора и дескриптора особых точек; число слов в словаре дескрипторов ключевых точек) осуществляется через аргументы командной строки.
2. Разбиение исходных данных на тренировочную и тестовую выборки случайным образом.
3. Выполнение следующей последовательности действий для тренировочной выборки:
   - нахождение ключевых точек и вычисление их дескрипторов;
   - обучение словаря;
   - построение признакового описания изображений;
   - формирование обучающей выборки и обучение классификатора «случайный лес».
4. Выполнение следующей последовательности действий для каждого изображения из тестовой выборки:
   - нахождение ключевых точек и вычисление их дескрипторов;
   - построение признакового описания;
   - предсказание категории изображения с использованием обученного классификатора и вычисленных признаков;
5. Вывод результатов классификации изображений на тестовой выборке.

3.1.2. Структура приложения

Приложение будет состоять из набора вспомогательных функций (auxiliary.cpp/h), основного модуля, содержащего реализацию основных стадий bag-of-words подхода к классификации изображений с использованием функций OpenCV (bow.cpp/h), и файла с исходным кодом основной функции (main.cpp). В рамках лабораторной работы предлагается реализовать функции из основного модуля; вспомогательные функции и код основной функции поставляется в реализованном виде.

Рассмотрим вспомогательные функции.

```
void GetFilesInFolder(const string& dirPath,
                      std::vector<string>& filesList)
```

Заполняет массив **filesList** списком всех файлов с расширением *.jpg из директории **dirPath**. Рассмотрим параметры данной функции:

- **dirPath** – путь к директории, содержащей изображения
- **filesList** – список всех файлов с расширением *.jpg, содержащихся в данной директории

```
void InitRandomBoolVector(vector<bool>& mask, double prob)
```

Заполняет булевский вектор **mask** случайными значениями (**true** с вероятностью **prob**). Рассмотрим параметры данной функции:

- **mask** – булевский вектор, который должен быть заполнен случайными значениями;
- **prob** – вероятность того, что элементу булевского массива будет присвоено значение **true**.

Рассмотрим функции из модуля bow.cpp, которые необходимо реализовать в рамках лабораторной работы.

```
Mat TrainVocabulary(const std::vector<string>& filesList,
            const std::vector<bool>& isVoc,
            const Ptr<FeatureDetector>& keypointsDetector,
            const Ptr<DescriptorExtractor>& dExtractor,
            int vocSize)
```

Функция возвращает построенный словарь дескрипторов особых точек и принимает на вход следующие аргументы:

- **filesList** – список файлов;
- **isVoc** – маска, описывающая набор изображений из **filesList**, используемых для построения словаря;
- **keypointsDetector** – детектор ключевых точек;

- **dExtractor** – алгоритм, используемый для вычисления дескрипторов ключевых точек;
- **vocSize** – размер словаря.

Функция должна выполнять следующую последовательность действий:

1. Создать объект класса **BOWTrainer** с размером словаря, равным vocSize и остальными параметрами по умолчанию.
2. Для каждого изображения из **filesList**, которое используется для построения словаря выполнить следующие действия:
   - Прочитать изображение из файла.
   - Продетектировать на изображении ключевые точки с использованием детектора **keypointsDetector**.
   - Вычислить дескрипторы ключевых точек с использованием алгоритма **dExtractor**.
   - Добавить вычисленные дескрипторы в набор дескрипторов, используемых объектом класса **BOWTrainer** для обучения словаря
3. Обучить словарь.

```
Mat ExtractFeaturesFromImage(
            Ptr<FeatureDetector> keypointsDetector,
            Ptr<BOWImgDescriptorExtractor> bowExtractor,
            const string& fileName)
```

Функция возвращает признаковое описание изображения и принимает на вход следующие аргументы:

- **keypointsDetector** – детектор ключевых точек;
- **bowExtractor** – алгоритм, используемый для вычисления признакового описания изображения;
- **fileName** – входной файл.

Функция должна выполнять следующую последовательность действий:

1. Прочитать изображение из файла **fileName**.
2. Продетектировать на изображении ключевые точки с использованием детектора **keypointsDetector**.
3. Вычислить признаковое описание изображения с использованием **bowExtractor**.

```
void ExtractTrainData(const std::vector<string>& filesList,
      const std::vector<bool>& isTrain,
      const Mat& responses,
      const Ptr<FeatureDetector>& keypointsDetector,
      const Ptr<BOWImgDescriptorExtractor>& bowExtractor,
      Mat& trainData,
      Mat& trainResponses)
```

Функция предназначена для формирования обучающей выборки (**trainData** и **trainResponses**) для используемого алгоритма обучения с учителем; принимает на вход следующие аргументы:

- **filesList** – список файлов;
- **isTrain** – маска, описывающая набор изображений из **filesList**, используемых для обучения классификатора;
- **responses** – ответы (категории) для файлов из **filesList**;
- **keypointsDetector** – детектор ключевых точек;
- **bowExtractor** – алгоритм, используемый для вычисления признакового описания изображения;
- **trainData** (выходной параметр) – матрица, содержащая признаковые описания изображений из обучающей выборки;
- **trainResponses** (выходной параметр) – матрица, содержащая ответы (категории) для изображений из обучающей выборки.

Функция должна выполнять следующую последовательность действий:

1. Вычислить число изображений в обучающей выборке (оно равно числу элементов в массиве **isTrain**, значение которых равно **true**).
2. Инициализировать матрицу **trainData** для хранения признакового описания изображений из обучающей выборки: число строк равно числу объектов в обучающей выборке, число столбцов равно размерности пространства признаков (числу слов в словаре дескрипторов особых точек, которое соответствует числу строк в словаре, который использует **bowExtractor**), элементами матрицы являются числа с плавающей запятой одинарной точности.
3. Инициализировать матрицу **trainResponses** для хранения ответов (категорий) объектов из обучающей выборки: число строк равно числу объектов в обучающей выборке, число столбцов равно 1, элементами матрицы являются 32-битные целые числа со знаком.
4. Для каждого изображения из **filesList**, которое используется для построения обучающей выборки:
   - Вычислить признаковое описание изображения.

- Скопировать вычисленное признаковое описание изображения в соответствующую строку матрицы **trainData**.
- Скопировать ответ (категорию) изображения в соответствующий элемент матрицы **trainResponses**.

```
Ptr<CvRTrees> TrainClassifier(const Mat& trainData,
                              const Mat& trainResponses)
```

Функция возвращает обученный классификатор «случайный лес» и принимает на вход следующие аргументы:

- **trainData** – матрица, содержащая признаковые описания изображений из обучающей выборки;
- **trainResponses** – матрица, содержащая ответы (категории) для изображений из обучающей выборки.

Функция должна выполнять следующую последовательность действий:

1. Создать структуру типа **CvRTParams**, описывающую параметры алгоритма обучения с учителем «случайный лес»:
   - Критерий остановки работы алгоритма – по числу итераций.
   - Максимальное число деревьев – 200.
2. Создать матрицу, описывающую типы входных переменных и ответа: число строк равно 1, число столбцов равно размерности пространства признаков (числу столбцов в матрице **trainData**) + 1, элементами матрицы являются 8-битные беззнаковые целые числа.
3. Установить типы всех используемых признаковых переменных и ответа: все признаковые переменные являются вещественными (**CV_VAR_ORDERED**), ответ является категориальным (**CV_VAR_CATEGORICAL**).
4. Создать объект класса **CvRTrees** (классификатор «случайный лес»).
5. Обучить классификатор.

```
float Predict(const Ptr<FeatureDetector> keypointsDetector,
      const Ptr<BOWImgDescriptorExtractor> bowExtractor,
      const Ptr<CvRTrees> classifier,
      const string& fileName)
```

Функция возвращает предсказанную категорию для изображения, хранящегося в файле **fileName**, и принимает на вход следующие параметры:

- **keypointsDetector** – детектор ключевых точек;

- **bowExtractor** – алгоритм, используемый для вычисления признакового описания изображения;
- **classifier** – обученный классификатор («случайный лес»);
- **fileName** – входной файл.

Функция должна выполнять следующую последовательность действий:

1. Вычислить признаковое описание изображения.
2. Предсказать категорию изображения с использованием обученного классификатора и вычисленного признакового описания.

```
Mat PredictOnTestData(const std::vector<string>& filesList,
      const std::vector<bool>& isTrain,
      const Ptr<FeatureDetector> keypointsDetector,
      const Ptr<BOWImgDescriptorExtractor> bowExtractor,
      const Ptr<CvRTrees> classifier)
```

Функция возвращает матрицу, содержащую предсказанные категории для всех объектов из тестовой выборки, и принимает на вход следующие аргументы:

- **filesList** – список файлов;
- **isTrain** – маска, описывающая набор изображений из **filesList**, используемых для обучения классификатора;
- **keypointsDetector** – детектор ключевых точек;
- **bowExtractor** – алгоритм, используемый для вычисления признакового описания изображения;
- **classifier** – обученный классификатор «случайный лес».

Функция должна выполнять следующую последовательность действий:

1. Вычислить число изображений в тестовой выборке (оно равно числу элементов в массиве **isTrain**, значение которых равно **false**).
2. Создать матрицу, используемую для хранения предсказанных категорий изображений: число строк равно числу объектов в тестовой выборке, число столбцов равно 1, элементами матрицы являются 32-битные целые числа со знаком.
3. Для каждого изображения из тестовой выборки: предсказать его категорию и скопировать её значение в соотвествующий элемент матрицы, хранящей предсказанные значения.

```
Mat GetTestResponses(const Mat& responses,
                     const vector<bool>& isTrain)
```

Функция возвращает матрицу, содержащую правильные категории (ответы) для изображений из тестовой выборки, и принимает на вход следующие аргументы:

- **`responses`** – ответы (категории) для всех изображений;
- **`isTrain`** – маска, описывающая набор изображений, использованных для обучения классификатора.

Функция должна выполнять следующую последовательность действий:

1. Вычислить число изображений в тестовой выборке (оно равно числу элементов в массиве **`isTrain`**, значение которых равно **`false`**).
2. Создать матрицу, используемую для хранения категорий изображений: число строк равно числу объектов в тестовой выборке, число столбцов равно 1, элементами матрицы являются 32-битные целые числа со знаком.
3. Для каждого объекта из тестовой выборки: скопировать категорию данного объекта в соотвествующий элемент матрицы, хранящей парвильные значения категорий объектов из тестовой выборки.

```
float CalculateMisclassificationError(Mat& responses,
                                      Mat& predictions)
```

Функция возвращает ошибку классификации (доля неправильно предсказанных категорий изображений) и принимает на вход следующие аргументы:

- **`responses`** – правильные ответы (категории);
- **`predictions`** – предсказанные ответы (категории).

Функция должна выполнять следующую последовательность действий:

1. Вычислить число несовпадений соответствующих элементов в массивах **`responses`** и **`predictions`**.
2. Вычислить отношение числа несовпадений к общему числу объектов.

Рассмотрим структуру основной функции приложения:

1. Считать аргументы, переданные программе через командную строку:
   - **`string folder1`** – путь к папке, содержащей объекты первой категории;

- **`string folder2`** – путь к папке, содержащей объекты второй категории;
- **`string detectorType`** – тип детектора ключевых точек;
- **`string descriptorType`** – тип дескрипторов ключевых точек;
- **`int vocSize`** – размер словаря;
- **`double trainProportion`** – доля объектов, используемых для построения словаря (и обучения классификатора «случайный лес»).

2. Инициализировать модуль nonfree, обеспечивающий работу с SIFT и SURF детекторами и дескрипторами.
3. Создать объект класса, детектирующего ключевые точки (типа **`detectorType`**).
4. Создать объект класса типа, вычисляющего дескрипторы ключевых точек (типа **`descriptorType`**).
5. Создать объект класса, предназначенного для нахождения ближайшего к дескриптору «слова» из словаря дескрипторов ключевых точек (типа «**`BruteForce`**»).
6. Создать объект класса, предназначенного для вычисления признакового описания изображений.
7. Создать массив, предназначенный для хранения списка, хранящего имена JPEG файлов из **`folder1`** и **`folder2`**.
8. Заполнить созданный список именами файлов из обеих категорий, вычислить число изображений, относящихся к первой и второй категории, и суммарное число изображений.
9. Создать массив для хранения маски, описывающей разбиение изображений на тренировочную и тестовую выборки. Инициализировать его случайными значениями таким образом, чтобы доля объектов, относящихся к тренировочной выборке была равна **`trainProportion`**.
10. Создать матрицу, содержащую категории изображений: число строк равно суммарному числу изображений, число столбцов равно 1, элементами матрицы являются 32-битные целые числа со знаком. Заполнить матрицу следующим образом: объектам, относящимся к первой категории, соотвествует значение 1, объектам, относящимся ко второй категории – -1.
11. Обучить словарь дескрипторов ключевых точек на изображениях, относящихся к тренировочной выборке.
12. Установить словарь.

13. Сформировать тренировочную выборку для классификатора «случайный лес».
14. Обучить классификатор «случайный лес» на сформированной выборке.
15. Предсказать категории изображений, относящихся к тестовой выборке.
16. Сформировать матрицу, содержащую правильные категории изображений из тестовой выборки.

    Вычислить и вывести ошибку классификации на тестовой выборке.

3.1.3. Параметры запуска приложения

Исходный вычислительный эксперимент необходимо провести со следующими параметрами:

- **`folder1`** – путь к папке crocodile;
- **`folder2`** – путь к папке leopards;
- **`detectorType`** – SIFT;
- **`descriptorType`** – SIFT;
- **`vocSize`** – 25;
- **`trainProportion`** – 0.5.

4. Контрольные вопросы

1. Какие основные этапы включают в себя bag-of-words алгоритмы классификации изображений?
2. Какие требования предъявляются к детекторам и дескрипторам ключевых точек?
3. Опишите идеи, лежащие в основе детектора и дескриптора SIFT.
4. Почему в качестве признакового описания изображения обычно не используются вычисленные дескрипторы ключевых точек?
5. Перечислите алгоритмы, которые возможно использовать при построении словаря дескрипторов ключевых точек.
6. Что представляет собой итоговое признаковое описание изображения в методах класса bag-of-words, и каким образом оно вычисляется?
7. Перечислите алгоритмы, которые можно использовать для классификации изображений с использованием вычисленного признакового описания.

5. Дополнительные задания

1. Добавьте в разработанное приложение вывод информации об изображениях из тестовой выборки, которые были неправильно классифицированы.
2. Добавьте в разработанное приложение возможность использования в качестве используемого классификатора машины опорных векторов с ядром типа Radial Basis Function. Сравните результаты с ранее реализованным подходом.
3. Выполните исследование зависимости ошибки классификации от используемых параметров (типа используемых детекторов и дескрипторов ключевых точек, числа слов в словаре, параметров алгоритма обучения с учителем «случайный лес»: числа деревьев в ансамбле, максимальной глубины деревьев, входящих в ансамбль).
4. Реализуйте построение словаря на основе Gaussian Mixture Model и сравните полученные результаты с ранее реализованным подходом.

6. Литература

6.1. Основная литература

1. Agrawal M., Konolige K., Blas M. Censure: Center surround extremas for realtime feature detection and matching. In European Conference on Computer Vision (ECCV 2008), pp. 102–115.
2. Bay H., Ess A., Tuytelaars T., Gool L.V. SURF: speed up robust features // Computer Vision and Image Understanding (CVIU). 2008. V.110, № 3. P. 346-359.
3. Bradski G., Kaehler A. Learning OpenCV Computer Vision with OpenCV Library. O' Reilly Media Publishers, 2008. 571p.
4. Calonder M., Lepetit V., Strecha C., Fua P. BRIEF: Binary Robust Independent Elementary Features // In Proceedings of the 11th European Conference on Computer Vision (ECCV'10). 2010.
5. Hastie T., Tibshirani R., Friedman J. The Elements of Statistical Learning. – Springer, 2008.
6. Ke Y., Sukthankar R. PCA-SIFT: A more distinctive representation for local image descriptors // In Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR'04). 2004. V.2. P. 506-513.
7. Lindeberg T. Feature detection with automatic scale selection // International Journal of Computer Vision. 1998. V.30. Issue 2.

8. Lowe D. Distinctive image features from scale-invariant keypoints // International Journal of Computer Vision. 2004. № 60. P. 91–110.
9. Matas J., Chum O., Urban M., PajdlaT. Robust wide baseline stereo from maximally stable extremal regions // British Machine Learning Conference. 2002. P. 384-393.
10. Mikolajczyk K., Schmid C. Scale and affine invariant interest point detectors // International Journal of Computer Vision. 2004. № 60(1). P. 63-86.
11. Rosten E., Drummond T. Machine Learning for high-speed corner detection // In Proceedings of the 9th European Conference on Computer Vision (ECCV'06). 2006. P. 430-443.
12. Rublee E., Rabaud V., Konolige K., Bradski G. ORB: an efficient alternative to SIFT or SURF // In Proceedings of the International Conference on Computer Vision (CVPR'11). 2011. P. 2564-2571
13. Szeliski R. Computer Vision: Algorithms and Applications. Springler, 2010. 979p.

6.2. Ресурсы сети Интернет

14. Caltech Datasets: object detection datasets [http://www.vision.caltech.edu/html-files/archive.html]
15. Formal Description of Moravec detector [http://kiwi.cs.dal.ca/~dparks/CornerDetection/moravec.htm].
16. Harris/Plessey Operator [http://kiwi.cs.dal.ca/~dparks/CornerDetection/harris.htm].
17. Документация модуля features2d библиотеки OpenCV [http://docs.opencv.org/modules/features2d/doc/common_interfaces_of_feature_detectors.html].
18. Документация модуля nonfree библиотеки OpenCV [http:// http://docs.opencv.org/modules/features2d/doc/common_interfaces_of_feature_detectors.html].
19. Официальная страница библиотеки OpenCV [http://opencv.org].
20. Схема работы детектора GoodFeaturesToTrack, представленная в документации OpenCV http://docs.opencv.org/modules/imgproc/doc/feature_detection.html].

7. Приложения

7.1. Приложение А. Исходный код основной функции консольного редактора изображений

```
#include "auxiliary.h"
#include "bow.h"
#include <opencv2/nonfree/nonfree.hpp>
#include <iostream>

using namespace cv;
using namespace std;

int main(int argc, char* argv[])
{
	/*
	TODO: реализовать функциональность,
	описаннную в разделе 3.1.2
	*/
}
```

7.2. Приложение Б. Исходный код заголовочного файла модуля, содержащего вспомогательные функции

```
#include <string>
#include <vector>

void GetFilesInFolder(const std::string& dirPath,
                  std::vector<std::string> &filesList);
void InitRandomBoolVector(std::vector<bool>& mask,
                          double prob);
```

7.3. Приложение В. Исходный код модуля, содержащего вспомогательные функции

```
#include "auxiliary.h"
#include <windows.h>
#include <opencv2/core/core.hpp>

using namespace std;
using namespace cv;

void	GetFilesInFolder(const	string&	dirPath,
std::vector<string> &filesList)
{
	HANDLE handle;
	WIN32_FIND_DATAA fileData;
```

```
      if     ((handle     =     FindFirstFileA((dirPath     +
"/*.jpg").c_str(), &fileData)) == INVALID_HANDLE_VALUE)
      {
            return;
      }
      do
      {
            const string file_name = fileData.cFileName;
            const string full_file_name = dirPath + "/" +
file_name;
            filesList.push_back(full_file_name);
      }
      while (FindNextFileA(handle, &fileData));
      FindClose(handle);
}

void InitRandomBoolVector(vector<bool>& mask, double prob)
{
      RNG rng = theRNG();
      for (size_t i = 0; i < mask.size(); i++)
      {
            mask[i] = (rng.uniform(0.0, 1.0) < prob) ? true
: false;
      }
}
```

7.4. Приложение Г. Исходный код заголовочного файла модуля, содержащего реализацию основных стадий bag-of-words подхода

```
#include <opencv2/features2d/features2d.hpp>
#include <opencv2/ml/ml.hpp>
#include <vector>
#include <string>

cv::Mat   TrainVocabulary(const   std::vector<std::string>&
filesList,   const   std::vector<bool>&   is_voc,   const
cv::Ptr<cv::FeatureDetector>&   keypointsDetector,   const
cv::Ptr<cv::DescriptorExtractor>& descriptorsExtractor, int
vocSize);

cv::Mat                            ExtractFeaturesFromImage(
cv::Ptr<cv::FeatureDetector>             keypointsDetector,
cv::Ptr<cv::BOWImgDescriptorExtractor>  bowExtractor,  const
std::string& fileName);

void    ExtractTrainData(const    std::vector<std::string>&
filesList, const std::vector<bool>& isTrain, const cv::Mat&
responses,
```

```
      const                    cv::Ptr<cv::FeatureDetector>&
keypointsDetector,                                        const
cv::Ptr<cv::BOWImgDescriptorExtractor>&       bowExtractor,
cv::Mat& trainData, cv::Mat& trainResponses);

cv::Ptr<CvRTrees> TrainClassifier(const cv::Mat& trainData,
const cv::Mat& trainResponses);
int        Predict(const        cv::Ptr<cv::FeatureDetector>
keypointsDetector,                                        const
cv::Ptr<cv::BOWImgDescriptorExtractor> bowExtractor,
      const     cv::Ptr<CvRTrees>     classifier,     const
std::string& fileName);

cv::Mat  PredictOnTestData(const  std::vector<std::string>&
filesList, const std::vector<bool>& isTrain,
      const cv::Ptr<cv::FeatureDetector> keypointsDetector,
const  cv::Ptr<cv::BOWImgDescriptorExtractor>  bowExtractor,
const cv::Ptr<CvRTrees> classifier);

cv::Mat  GetTestResponses(const  cv::Mat&  responses,  const
std::vector<bool>& isTrain);

float  CalculateMisclassificationError(cv::Mat&  responses,
cv::Mat& predictions);
```

7.5. Приложение Д. Исходный код модуля, содержащего реализацию основных стадий bag-of-words подхода

```
#include "bow.h"
#include <opencv2/highgui/highgui.hpp>

using namespace cv;
using namespace std;

Mat TrainVocabulary(const vector<string>& filesList, const
vector<bool>& is_voc,
      const Ptr<FeatureDetector>& keypointsDetector, const
Ptr<DescriptorExtractor>&     descriptorsExtractor,     int
vocSize)
{
      /*
      TODO: реализовать функциональность,
      описаннную в разделе 3.1.2
      */

}

Mat ExtractFeaturesFromImage(
            Ptr<FeatureDetector> keypointsDetector,
            Ptr<BOWImgDescriptorExtractor> bowExtractor,
```

```
            const string& fileName)
{
      /*
      TODO: реализовать функциональность,
      описаннную в разделе 3.1.2
      */
}

void ExtractTrainData(const vector<string>& filesList,
      const vector<bool>& isTrain,
      const Mat& responses,
      const Ptr<FeatureDetector>& keypointsDetector,
      const Ptr<BOWImgDescriptorExtractor>& bowExtractor,
      Mat& trainData,
      Mat& trainResponses)
{
      /*
      TODO: реализовать функциональность,
      описаннную в разделе 3.1.2
      */
}

Ptr<CvRTrees> TrainClassifier(const Mat& trainData,
                        const Mat& trainResponses)
{
      /*
      TODO: реализовать функциональность,
      описаннную в разделе 3.1.2
      */
}

int Predict(const Ptr<FeatureDetector> keypointsDetector,
      const Ptr<BOWImgDescriptorExtractor> bowExtractor,
      const Ptr<CvRTrees> classifier,
      const string& fileName)
{
      /*
      TODO: реализовать функциональность,
      описаннную в разделе 3.1.2
      */
}

Mat PredictOnTestData(const vector<string>& filesList,
      const vector<bool>& isTrain,
      const Ptr<FeatureDetector> keypointsDetector,
      const Ptr<BOWImgDescriptorExtractor> bowExtractor,
      const Ptr<CvRTrees> classifier)
{
            /*
      TODO: реализовать функциональность,
```

```
      описаннную в разделе 3.1.2
      */
}

Mat GetTestResponses(const Mat& responses,
                  const vector<bool>& isTrain)
{
      /*
      TODO: реализовать функциональность,
      описаннную в разделе 3.1.2
      */
}

float CalculateMisclassificationError(Mat& responses,
                                      Mat& predictions)
{
      /*
      TODO: реализовать функциональность,
      описаннную в разделе 3.1.2
      */
}
```