**5** ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

Тестирование приложения является неотъемлемой частью разработки, а также одним из важнейших шагов на пути реализации того или иного функционала в программе. Тестирование программного обеспечения позволяет установить соответствие между реальным поведением программы, и ожидаемым.

**5.1** Верификация нейронной сети

Для верификации нейронной сети необходимо определить критерии оценки результата колоризации. Среди различных критериев оценки можно выделить следующие:

* соответствие результата колоризации оригинальному изображению, до его преобразования в полутоновое (при наличии такового);
* субъективное мнение пользователей; результат колоризации может не соответствовать действительности, но при этом быть приемлемым для пользователя (пример на рисунке 5.1).
* 
* Рисунок 5.1 – Пример ошибочной, но приемлемой колоризации
* Несмотря на то, что субъективное мнение пользователей, является неплохой оценкой для работы нейронной сети, этот критерий не подходит для данного дипломного проекта. Главным недостатком является тот факт, что для того, чтобы получить адекватную оценку, необходимо набрать достаточное количество уникальных пользователей клиентского приложения, что невозможно без его выпуска, на стадии тестирования. Также стоит заметить, что для использования данного критерия оценки необходимо дополнительно реализовать пользовательский интерфейс, для назначения оценки, что не было запланировано на этапе функционального проектирования.
* Соответствие результата колоризации исходному изображению до преобразования в полутоновое больше подходит для этапа тестирования продукта. Для того, чтобы оценка приемлемой, вместо относительно большого количества уникальных пользователей приложения необходимо достаточное количество запросов на колоризацию изображений. Несмотря на тот факт, что для реализации верификации данным способом не требуется построение дополнительного интерфейса для пользователя, необходим дополнительный модуль для оценки колоризации на клиентском приложении. Дополнительный модуль должен реализовывать алгоритм, принимающий исходное изображение, не подверженное никакой обработке, результат колоризации. В качестве результата алгоритм должен возвращать оценку результату колоризации.
* Для данной цели был создан размещённый в одноимённом файле класс ColorizationEvaluator, который является частью java-пакета utils клиентского приложения. Данный класс имеет только статические методы и поля, что упрощает его интеграцию в разные части приложения:

public class ColorizationEvaluator {  
  
 private static Bitmap original;  
 private static Bitmap result;  
  
 public static void setOriginal(Bitmap original) {  
 ColorizationEvaluator.original = original;  
 }  
  
 public static void setResult(Bitmap result) {  
 ColorizationEvaluator.result = result;  
 }  
  
 public static Float getEvaluation() {  
 if (original == null || result == null  
 || original.getHeight() != result.getHeight()  
 || original.getWidth() != result.getWidth())  
 return null;  
 int height = original.getHeight();  
 int width = original.getWidth();  
 float evaluationSum = 0;  
 for (int i = 0; i < width; i++)  
 for (int j = 0; j < height; j++) {  
 int originalPixel = original.getPixel(i, j);  
 int resultPixel = result.getPixel(i, j);  
 int originalR = getRChannel(originalPixel);  
 int resultR = getRChannel(resultPixel);  
 int originalG = getGChannel(originalPixel);  
 int resultG = getGChannel(resultPixel);  
 int originalB = getBChannel(originalPixel);  
 int resultB = getBChannel(resultPixel);  
 evaluationSum += getEvaluationForChannel(originalR, resultR);  
 evaluationSum += getEvaluationForChannel(originalG, resultG);  
 evaluationSum += getEvaluationForChannel(originalB, resultB);  
 }  
 return evaluationSum / (height \* width \* 3);  
 }  
  
 private static float getEvaluationForChannel(int original, int result) {  
 float maxDelta = original > 127 ? original : 255 - original;  
 float error = Math.abs(result - original);  
 return (maxDelta - error) / maxDelta;  
 }  
  
 private static int getRChannel(int pixel) {  
 return (pixel & 0x00FF0000) >> 16;  
 }  
  
 private static int getGChannel(int pixel) {  
 return (pixel & 0x0000FF00) >> 8;  
 }  
  
 private static int getBChannel(int pixel) {  
 return (pixel & 0x000000FF);  
 }  
  
 public static void clearBitmaps() {  
 original = null;  
 result = null;  
 }  
}

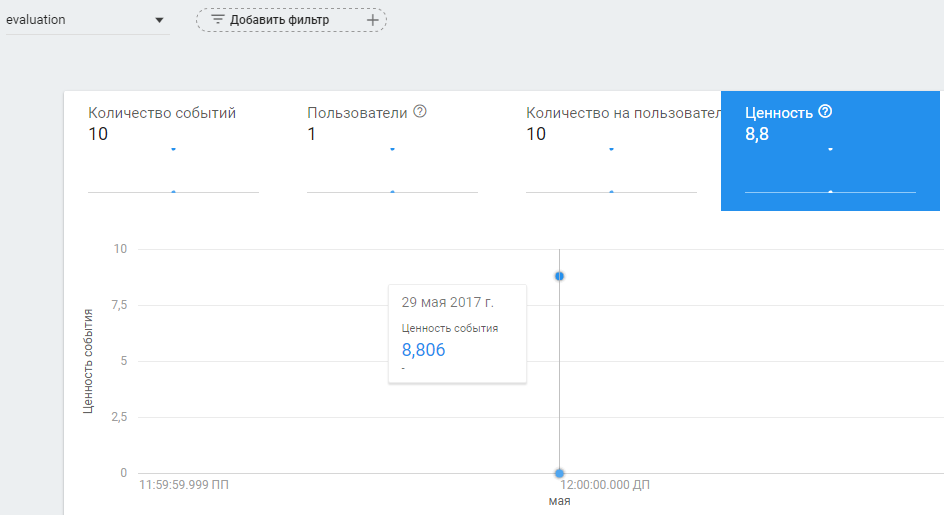
* Данный класс содержит следующие методы:
* setOriginal – метод для сохранения исходного изображения до его преобразования в полутон; вызывается классом GetBitmapTask после изменения разрешения изображения;
* setResult – метод для сохранения результата колоризации; вызывается классом DownloadingTask после декодирования изображения из строки формата Base64;
* getEvaluation – метод, возвращающий среднюю оценку точности колоризации пикселя по всему изображению. Именно данная величина используется в качестве оценки для колоризации;
* getEvaluationForChannel – инкапсулированный метод, возвращающий оценку точности колоризации для отдельного канала определённого пикселя;
* getRChannel – инкапсулированный метод, возвращающий значение канала, ответственного за наличие красного цвета; использует бинарные операторы;
* getGChannel – инкапсулированный метод, возвращающий значение канала, ответственного за наличие зелёного цвета; использует бинарные операторы;
* getBChannel – инкапсулированный метод, возвращающий значение канала, ответственного за наличие зелёного цвета; использует бинарные операторы;
* clearBitmaps – метод для очистки ссылок на Bitmap-объекты; вызывается после получения оценки классом DownloadingTask.
* В качестве оценки точности колоризации изображения используется среднее значение точности колоризации отдельно взятых каналов изображения. Для одного канала, ответственного за наличие цветовой составляющей пикселя, оценка вычисляется как отношение модуля разности оригинального значения и результата колоризации к максимально возможной разности двух каналов.
* Для сбора данных используется Firebase Analytics. Так как Firebase Cloud Messaging уже был имплементирован в данный проект, установка сбора данных с помощью Firebase Analytics не потребовало больших временных затрат: достаточно лишь добавить дополнительную зависимость в файл build.gradle:

dependencies {  
 // ...  
 compile 'com.google.firebase:firebase-core:10.2.4'  
 // ...  
}

* Для отправки новой величины оценки колоризации, необходимо поместить её в объект типа Bundle c ключом VALUE и передать в метод сущности FirebaseAnalytics:

Bundle report = new Bundle();  
report.putFloat(FirebaseAnalytics.Param.VALUE, evaluation);  
FirebaseAnalytics.getInstance(activity)  
 .logEvent(EVENT\_NAME, report);

* Для просмотра статистики необходимо перейти в Firebase Console. При выбранном диапазоне дат будут отображены количество событий с определённым именем и суммарная VALUE от их.



* Рисунок 5.2 – Фрагмент статистики от Firebase Analytics

Так, например, для первых десяти колоризаций средняя точность составила 88,06% (рисунок 5.2). Недостатком данной статистики является, её время обновления. Максимальное время, через которое возможно обновление статистики в веб-приложении составляет 24 часа по заявлению разработчиков. При получении данных для проекта задержка была равна около 8 часам.

На основании полученных данных средняя величина колоризации составляет 67.8%. Данная оценка составлена на основании 126 результатов колоризации изображений различных объектов. Стоит отметить, что нейронная сеть лучше справляется с колоризацией отдельных объектов, имеющих определённый характерный цвет, текстур дерева, растительности. Худшие результаты даёт колоризация фотографий, имеющих большое количество мелких деталей, колоризация объектов, имеющих редкий цвет.

**5.2** Затраты времени на колоризацию

* Для измерения затрат времени на колоризацию производились замеры времени работы метода colorize на стороне сервера. Во процессе экспериментальных замеров времени был установлен факт: время колоризации пропорционально разрешению исходного изображения. Поэтому в дальнейшем было решено производить замеры времени только на изображениях, сделанных с помощью камеры с разрешением матрицы 3120x4160, уменьшенных в процессе предобработки в 4 раза до разрешения 780x1040. На основании двадцати колоризаций было вычислено среднее её время, равное 4.67 секундам. Наибольшее и наименьшее затраченное на колоризацию время составило соответственно 4.8 и 4.57 секунды.
* Затраты времени на загрузку и выгрузку изображений, посылку-приём GCM-сообщения не были измерены, так как данные величины сильно зависят от типа подключения и состояния сети в определённый момент времени.
* **5.3** Ручное функциональное тестирование приложения
* После реализации клиентского приложения было проведено функциональное тестирование приложения
* Функциональные тесты, проведенные над разработанным приложением, представлены в таблице 5.1.

Таблица 5.1 ­ Тестирование программы

|  |  |  |  |
| --- | --- | --- | --- |
| Модуль | Содержание теста | Ожидаемый результат | Тест пройден |
| 1 | 2 | 3 | 4 |
| Контроллеров | Авторизация в системе после ввода корректных пользовательских данных | Пользователь авторизован в системе, на форме отрисован его логин | Да |
| Контроллеров | Авторизация в системе после ввода некорректных пользовательских данных | Пользователь не авторизован, выполнен переход на начальную страницу приложения | Да |
| Контроллеров | Выход из системы | Пользователь не авторизован, если в этот момент пользователь находился на одной из фом ­ переход на начальную страницу приложения | Да |
| Контроллеров | Переход на страницу авторизации | Открыта страница авторизации | Да |
| Контроллеров | Переход на страницу с расписание группы по определенному предмету с вводом валидного номера группы | Открыта страница с расписанием, проставлены оценки студентов по определённым занятиям | Да |

*Продолжение таблицы 5.1*

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| Контроллеров | Переход на страницу с расписание группы по определенному предмету с вводом невалидного номера группы | Кнопка, осуществляющая переход, неактивна | Да |
| Контроллеров | Переход на страницу добавления студента | Открыта страница с формой для заполнения данных о студенте | Да |
| Контроллеров | Переход на страницу добавления занятия | Открыта страница с формой для заполнения данных о занятии | Да |
| Контроллеров | Переход на страницу добавления предмета | Открыта страница с формой для заполнения данных о предмете | Да |
| Контроллеров | Переход на страницу добавления аудитории | Открыта страница с формой для заполнения данных об аудитории | Да |
| Связи с базой данных | Добавление нового студента | Информация о новом студенте добавлена в базу данных | Да |
| Связи с базой данных | Добавление нового занятия группе | Информация о новом занятии добавлена в базу данных | Да |
| Связи с базой данных | Добавление нового предмета | Информация о новом предмете добавлена в базу данных | Да |
| Связи с базой данных | Добавление новой аудитории | Информация об аудитории добавлена в базу данных | Да |
| Отображения | Отображение выпадающего списка в возможными действиями по нажатию кнопки «Навигация» пользователем не являющимся администратором | Отображен выпадающий список с формой для перехода на страницу расписания, а так же кнопкой по переходу на начальную страницу приложения | Да |

*Продолжение таблицы 5.1*

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| Отображения | Отображение выпадающего списка в возможными действиями по нажатию кнопки «Навигация» пользователем являющимся администратором | Отображен выпадающий список с возможностями обычных пользователей, а так же набором кнопок для перехода на страницы с формами по добавлению новых сущностей | Да |
| Отображения | Отображение выпадающего списка с формой для авторизации по нажатию кнопки «Войти» | Отображен выпадающий список с формой для ввода пользовательских данных и кнопкой «Войти» | Да |