**4** РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

В данном разделе будет рассмотрена реализация отдельных блоков приложения. Для реализации клиентского приложения для платформы Android использовались среда разработки Android Studio и язык программирования Java. Также была использована система автоматической сборки Gradle, встроенная в Android Studio, в качестве основного модуля сборки проекта. Для реализации серверной части были использованы:

* язык программирования Python;
* интерпретатор языка Python версии 2.7;
* среда разработки JetBrains PyCharm Community Edition, которая имеет интерфейс схожий с Android Studio и позволяет проверить синтаксис скрипта на языке Python;
* текстовый редактор Sublime Text 3, который использовался в качестве более удобного способа изменения источников кода.

Распределённая система управления версиями Git использовалась, для хранения версий источников кода и клиентского приложения, и серверной части. Стоит также отметить, что Git позволила более удобно передавать исходный код внутрь Docker-контейнера, что крайне затруднительно при использовании Windows-хоста, из-за различия в структурах файловых систем.

**4.1** Обучение нейронной сети

Для обучения нейронной сети необходимы:

* База изображений в формате LMDB файлов;
* Видеокарта, для ускорения процесса обучения;
* Время, для исполнения достаточного количества итераций обучения.

Согласно статье [9] для подобной сети требуется около 450000 итераций, что при отсутствии видеокарты, может занимать около от 55 до 90 часов обучения при использовании фреймворка Caffe. Ввиду ограниченного количества времени на реализацию проекта, отсутствия графического процессора на персональном компьютере, места для хранения базы изображений, а также стабильного Интернет-соединения для загрузки базы изображений, в готовом проекте использована предобученная нейронная сеть. Однако, в данном подразделе всё же будет описан процесс обучения нейронной сети на основе информации из первоисточника [10].

Для обучения нейронной сети необходимо выгрузить некоторые файлы из удалённого git-репозитория первоисточника. Все команды, которые необходимо выполнить в процессе обучения вводятся с помощью интерфейса командной строки Linux-системы.

wget eecs.berkeley.edu/~rich.zhang/projects/2016\_colorization/files/train/init\_v2.caffemodel -O ./init.caffemodel

Исполнение данной команды загрузит Caffe-модель, которую инициализировали методом k-средних, в корневой каталог выгруженного проекта.

wget eecs.berkeley.edu/~rich.zhang/projects/2016\_colorization/files/train/caffe-colorization.tar.gz -O ./caffe-colorization.tar.gz

tar -xvf ./caffe-colorization.tar.gz

rm ./caffe-colorization.tar.gz

Выполнение данных команд загрузит модифицированный под задачу обучения конкретной нейронной сети фреймворк Caffe, упакованный в архив с расширением tar.gz. Затем произойдёт распаковка архива и его удаление.

Затем следует добавить каталог resources проекта в системную переменную $PYTHONPATH. Данный шаг может различаться для разных дистрибутивов Linux. Здесь приведён способ добавить системную переменную в операционной системе Ubuntu:

export VAR\_NAME=VAR\_VALUE

После добавления системной переменной необходимо переопределить пути к базе изображений в файле models/colorization\_train\_val\_v2.prototxt. Затем само обучение запускается при помощи следующей команды:

./caffe-colorization/build/tools/caffe train -solver ./train/solver.prototxt -weights ./init.caffemodel -gpu $1

Следующая команда начнёт обучение модели. В процессе обучения будет использована видеокарта, идентификатор которой указан флагом gpu. Файл solver.prototxt содержит следующие параметры:

net: "./models/colorization\_train\_val\_v2.prototxt"

test\_iter: 1 # Количество итераций за шаг

test\_interval: 10000000

test\_initialization: False

base\_lr: 3.16e-5

lr\_policy: "step"

gamma: 0.316

stepsize: 215000

display: 1

max\_iter: 500000

momentum: 0.9

momentum2: 0.99

weight\_decay: 0.001

snapshot: 1000

snapshot\_prefix: "./train/models/colornet"

solver\_mode: GPU

type: "Adam"

average\_loss: 1000

Обучение заканчивается после 500000 итераций. Само обучение происходит на случайных частях изображений, которые также могут быть зеркальными по отношению к оригиналу. Снимки, отображающие процесс обучения, будут сохраняться каждые 1000 итерации по пути, который указан в параметре snapshot\_prefix файла solver. Сами снимки имеют расширение solverstate.

**4.2** Описание архитектуры сети

В данном подразделе указана часть файла deploy.prototxt, который находится в каталоге neuralnet проекта и используется модулем colorization.py. Данный файл содержит описание архитектуры свёрточной нейронной сети.

name: "LtoAB"

layer {

name: "data\_l"

type: "Input"

top: "data\_l"

input\_param {

shape { dim: 1 dim: 1 dim: 224 dim: 224 }

}

}

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# \*\*\*\*\* conv1 \*\*\*\*\*

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

layer {

name: "bw\_conv1\_1"

type: "Convolution"

bottom: "data\_l"

top: "conv1\_1"

convolution\_param {

num\_output: 64

pad: 1

kernel\_size: 3

}

}

layer {

name: "relu1\_1"

type: "ReLU"

bottom: "conv1\_1"

top: "conv1\_1"

}

layer {

name: "conv1\_2"

type: "Convolution"

bottom: "conv1\_1"

top: "conv1\_2"

convolution\_param {

num\_output: 64

pad: 1

kernel\_size: 3

stride: 2

}

}

layer {

name: "relu1\_2"

type: "ReLU"

bottom: "conv1\_2"

top: "conv1\_2"

}

layer {

name: "conv1\_2norm"

type: "BatchNorm"

bottom: "conv1\_2"

top: "conv1\_2norm"

batch\_norm\_param{ }

param {lr\_mult: 0 decay\_mult: 0}

param {lr\_mult: 0 decay\_mult: 0}

param {lr\_mult: 0 decay\_mult: 0}

}

В следующем примере указаны метка-имя архитектуры name, входной слой с именем data\_l, а также составной слой conv\_1, состоящий из 5 слоёв разных типов:

* Convolution – слой свёртки; пример такого слоя с некоторыми его параметрами был описан в разделе функционального проектирования;
* ReLU – слой функции активации после свёрточного слоя; для активации выбирается вместо обычных функций типа гиперболического тангенса или сигмоиды ненасыщаемая функция, которая принимает значение аргумента, если сам аргумент больше нуля, и значение нуля, если аргумент – отрицательное число. Такая функция показывает хорошие результаты при обучении нейронных сетей и отвечает за отсечение ненужных деталей в канале (при отрицательном выходе);
* BatchNorm – слой батч-нормализации.

Из параметров к слою свёртки, кроме уже описанных, здесь приведён параметр stride, обозначающий интервал в пикселях, на который сдвигается маска. Наличие слоёв батч-нормализации связано со следующей проблемой: по мере распространения сигнала по сети, даже если нормализовывать его на входе, пройдя через внутренние слои, сигнал может сильно исказиться как по математическому ожиданию, так и по дисперсии (данное явление называется внутренним ковариационным сдвигом), что чревато серьезными несоответствиями между градиентами на различных уровнях.

Одним из последних слоёв нейронной сети является слой Softmax. Особенностью данного слоя является тот факт, что сумма всех значений выходов нейронов этого слоя равна единице:

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# \*\*\*\*\* Softmax \*\*\*\*\*

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

layer {

name: "conv8\_313"

type: "Convolution"

bottom: "conv8\_3"

top: "conv8\_313"

convolution\_param {

num\_output: 313

kernel\_size: 1

stride: 1

dilation: 1

}

}

layer {

name: "conv8\_313\_rh"

type: "Scale"

bottom: "conv8\_313"

top: "conv8\_313\_rh"

scale\_param {

bias\_term: false

filler { type: 'constant' value: 2.606 }

}

}

layer {

name: "class8\_313\_rh"

type: "Softmax"

bottom: "conv8\_313\_rh"

top: "class8\_313\_rh"

}

Последним является слой для декодирования значений результата предыдущего слоя в значения каналов A и B:

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# \*\*\*\*\* Decoding \*\*\*\*\*

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

layer {

name: "class8\_ab"

type: "Convolution"

bottom: "class8\_313\_rh"

top: "class8\_ab"

convolution\_param {

num\_output: 2

kernel\_size: 1

stride: 1

dilation: 1

}

}

layer {

name: "Silence"

type: "Silence"

bottom: "class8\_ab"

}

Именно с выходов слоя class8\_ab снимается значения цветов пикселей. Cлой Silence добавляется в качестве заглушки для готовых моделей сети.

**4.3** Модуль colorization.py

В данном подразделе будет в основном описано использование Python-интерфейса фреймворка Caffe.

Выбор центрального процессора в качестве устройства для вычисления значений при прямом проходе по сети:

caffe.set\_mode\_cpu()

Загрузка модели нейронной сети:

net = caffe.Net(prototxt, caffemodel, caffe.TEST)

Аргументы метода Net:

* prototxt – путь к prototxt-файлу, описывающему архитектуру сети;
* caffemodel – путь к caffemodel-файлу, содержащему обученную модель сети;
* caffe.TEST – значение, которое указывает на необходимость вывода отчёта о работе сети в поток STDOUT.

Получение размеров входного и выходного слоя нейронной сети:

(H\_in, W\_in) = net.blobs['data\_l'].data.shape[2:]

(H\_out, W\_out) = net.blobs['class8\_ab'].data.shape[2:]

Загрузка изображения из файловой системы:

img\_rgb = caffe.io.load\_image(img\_in)

Получение одноканального изображения (L-канал) для последующей конкатенации с результатом нейронной сети:

img\_lab = color.rgb2lab(img\_rgb)

img\_l = img\_lab[:, :, 0]

Получение размеров исходного изображения:

(H\_orig, W\_orig) = img\_rgb.shape[:2]

Получение одноканального изображения (L-канал) для входного слоя нейронной сети (изменения размеров исходного изображения для входного слоя, перевод в цветовую схему LAB и выделения канала яркости:

img\_rs = caffe.io.resize\_image(img\_rgb, (H\_in, W\_in))

img\_lab\_rs = color.rgb2lab(img\_rs)

img\_l\_rs = img\_lab\_rs[:, :, 0]

Загрузка исходного изображения в сеть:

net.blobs['data\_l'].data[0, 0, :, :] = img\_l\_rs - 50

Запуск нейронной сети:

net.forward()

Снятие результата с выходного слоя сети:

ab\_dec = net.blobs['class8\_ab'].data[0, :, :, :].transpose((1, 2, 0))

Увеличение разрешения результата до разрешения оригинального изображения:

ab\_dec\_us = sni.zoom(ab\_dec, (1.\*H\_orig / H\_out, 1.\*W\_orig / W\_out, 1))

Получение обработанного изображения в цветовой модели LAB и последующий перевод в цветовую модель RGB:

img\_lab\_out = np.concatenate((img\_l[:, :, np.newaxis], ab\_dec\_us), axis = 2)

img\_rgb\_out = (255 \* np.clip(color.lab2rgb(img\_lab\_out), 0, 1)).astype('uint8')

Сохранение изображения в файловую систему:

plt.imsave(img\_out, img\_rgb\_out)

Вызов модуля оповещения:

notifier.notify(id, token)

Весь код модуля управления нейронной сетью приложен в листинге.

**4.4** GCM-модуль оповещения:

В данном подразделе указан код GCM-модуля оповещения целиком:

from gcm import GCM

def notify(id, token):

gcm = GCM("AIzaSyBk6qJ9tIij0GCezqDUfGu-IekXmbayE2k")

data = {'id': id}

gcm.plaintext\_request(registration\_id=token, data=data)

Функция GCM возвращает объект, для дальнейшей работы с GCM библиотекой. Принимает она API-ключ, который был получен при регистрации проекта в веб-приложении Firebase. Объект data заключает в себе данные, которые будут переданы через GCM-сообщение. Вызов функции plaintext\_request инициализирует отправку сообщения клиенту с идентификатором token.

**4.5** Активность для выбора исходного изображения

В данном подразделе будет указан код файла PickerActivity.java, который является входной точкой приложения. При создании активности вызывается метод onCreate, в котором происходит инициализация интерфейса пользователя:

@Override  
protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_picker);  
 findViews();  
 setUpViews();  
}  
  
private void findViews() {  
 imageView = (ImageView) findViewById(R.id.iv);  
 progressBar = (ProgressBar) findViewById(R.id.pb);  
 cameraButton = (Button) findViewById(R.id.b\_camera);  
 galleryButton = (Button) findViewById(R.id.b\_gallery);  
 loadButton = (Button) findViewById(R.id.b\_load);  
}  
  
private void setUpViews() {  
 cameraButton.setOnClickListener(this);  
 galleryButton.setOnClickListener(this);  
 loadButton.setOnClickListener(this);  
}

Метод findViewById связывает объект интерфейса с ссылкой на него в активности. Метод setOnClickListener объявляет обработчиком нажатий на контроллеры объект активности, который реализует интерфейс View.OnClickListener:

@Override  
public void onClick(View v) {  
 enableControls(false);  
 switch (v.getId()) {  
 case R.id.b\_camera:  
 Pair<Intent, Uri> pair = CameraProvider.getCameraIntent(this);  
 imageUri = pair.second;  
 startActivityForResult(pair.first, CODE\_REQUEST\_CAMERA);  
 break;  
 case R.id.b\_gallery:  
 Intent pickPhoto = new Intent(Intent.ACTION\_PICK,  
 android.provider.MediaStore.Images.Media.EXTERNAL\_CONTENT\_URI);  
 startActivityForResult(pickPhoto, CODE\_REQUEST\_GALLERY);  
 break;  
 case R.id.b\_load:  
 if (bitmap != null) {  
 BitmapHolder.save(bitmap);  
 startActivity(new Intent(this, ColorizationActivity.class));  
 }  
 enableControls(true);  
 break;  
 }  
}

При нажатии на объект интерфейса с идентификатором R.id.b\_load вызывается метод startActivity, который в свою очередь вызывает активность для передачи изображения на сервер и отображение обработанного изображения. При нажатии на другие элементы интерфейса вызывается метод startActivityForResult, который в свою очередь вызывает стороннюю активность, возвращающую URI выбранного изображения:

@Override  
protected void onActivityResult(int requestCode, int resultCode, Intent data) {  
 super.onActivityResult(requestCode, resultCode, data);  
 if (resultCode == RESULT\_OK) {  
 Uri uri = requestCode == CODE\_REQUEST\_CAMERA ? imageUri  
 : requestCode == CODE\_REQUEST\_GALLERY ? data.getData() : null;  
 imageUri = null;  
 if (uri != null) {  
 AsyncTaskCompat.executeParallel(new GetBitmapTask(this, uri) {  
 @Override  
 protected void onPostExecute(Throwable tr) {  
 super.onPostExecute(tr);  
 enableControls(true);  
 if (tr == null) {  
 PickerActivity.this.bitmap = getBitmap();  
 imageView.setImageBitmap(bitmap);  
 } else Toast.makeText(PickerActivity.this, tr.getMessage(),  
 Toast.LENGTH\_SHORT).show();  
 }  
 });  
 return;  
 }  
 }  
 enableControls(true);  
}

При получении URI от сторонней активности вызывается метод AsyncTaskCompat.executeParallel, который запускает объект AsyncTask, переданный в метод в качестве параметра. При получении изображения вызывается метод ImageView.setImageBitmap, который отображает полученное изображение на экран.

**4.6** Модуль предобработки изображения

В данном подразделе описан код файла ImageProcessor.java, предоставляющий интерфейс для обработки изображений:

public static Bitmap getGrayScale(Bitmap original) {  
 int width, height;  
 height = original.getHeight();  
 width = original.getWidth();  
 Bitmap grayscale = Bitmap.createBitmap(width, height, Bitmap.Config.ARGB\_8888);  
 Canvas c = new Canvas(grayscale);  
 Paint paint = new Paint();  
 ColorMatrix cm = new ColorMatrix();  
 cm.setSaturation(0);  
 ColorMatrixColorFilter f = new ColorMatrixColorFilter(cm);  
 paint.setColorFilter(f);  
 c.drawBitmap(original, 0, 0, paint);  
 return grayscale;  
}  
  
public static Bitmap resize(Bitmap original, float scale) {  
 int width = original.getWidth();  
 int height = original.getHeight();  
 int newWidth = (int) (width \* scale);  
 int newHeight = (int) (height \* scale);  
 float scaleWidth = ((float) newWidth) / width;  
 float scaleHeight = ((float) newHeight) / height;  
 Matrix matrix = new Matrix();  
 matrix.postScale(scaleWidth, scaleHeight);  
 return Bitmap.createBitmap(original, 0, 0, width, height, matrix, false);  
}

Для получения полутонового изображения используется метод getGrayScale. Для создания нового Bitmap-объекта используется метод Bitmap.createBitmap. Для того, чтобы получить значения яркости изображения используется объект ColorMatrix, методы ColorMatrix.setSaturation, и Canvas.drawBitmap.

В методе resize cначала вычисляются scaleWidth и scaleHeight обработанного изображения. Затем с помощью объекта matrix и метода createBitmap можно получить изображение с новым разрешением.

**4.7** Активность для отображения колоризованного изображения

При создании активности вызывается метод onCreate, в котором происходит получение изображения из BitmapHolder, отображение полученного изображения на экран, регистрация объекта PushHandler и вызов AsyncTask.executeParallel на объекте UploadingTask:

@Override  
protected void onCreate(@Nullable Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_colorization);  
 findViews();  
 backButton.setOnClickListener(this);  
 bitmap = BitmapHolder.get();  
 imageView.setImageBitmap(bitmap);  
 handler = new PushHandler(this);  
 LocalBroadcastManager.getInstance(this).registerReceiver(handler,  
 new IntentFilter(INTENT\_ACTION));  
 currentTask = new UploadingTask(bitmap) {  
 @Override  
 protected void onPreExecute() {  
 super.onPreExecute();  
 progressBar.setVisibility(View.VISIBLE);  
 }  
  
 @Override  
 protected void onPostExecute(Throwable tr) {  
 super.onPostExecute(tr);  
 if (tr != null) progressBar.setVisibility(View.GONE);  
 Toast.makeText(ColorizationActivity.this, tr == null ? LOADING\_SUCCESS\_MESSAGE  
 : tr.getMessage(), Toast.LENGTH\_SHORT).show();  
 }  
 };  
 AsyncTaskCompat.executeParallel(currentTask);  
}

При уничтожении активности вызывается метод onDestroy, внутри которого происходит отмена регистрации объекта PushHandler:

@Override  
protected void onDestroy() {  
 LocalBroadcastManager.getInstance(this).unregisterReceiver(handler);  
 super.onDestroy();  
}

В случае нажатия на кнопку «BACK» или на соответствующий контроллер смартфона вызывается отмена текущего AsyncTask-объекта:

@Override  
public void onClick(View v) {  
 switch (v.getId()) {  
 case R.id.b\_back:  
 if (currentTask != null && !currentTask.isCancelled())  
 currentTask.cancel(true);  
 finish();  
 break;  
 }  
}  
@Override  
public void onBackPressed() {  
 if (currentTask != null && !currentTask.isCancelled())  
 currentTask.cancel(true);  
 super.onBackPressed();  
}

**4.7** Задача загрузки изображения

В данном подразделе приведён код файла UploadingTask.java. В данном файле кроме посылки запроса на сервер заключены перевод в формат изображения в строку формата Base64 и назначения идентификаторов изображениям:

@Override  
protected final Throwable doInBackground(Object... params) {  
 try {  
 String base64 = getBase64(bitmap);  
 String id = getRandomId();  
 API.getInterface().loadImage(FirebaseInstanceId.getInstance().getToken(), id, base64)  
 .execute();  
 return null;  
 } catch (IOException e) {  
 return e;  
 }  
}  
  
private String getBase64(Bitmap original) {  
 ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();  
 original.compress(Bitmap.CompressFormat.PNG, 100, byteArrayOutputStream);  
 byte[] byteArray = byteArrayOutputStream.toByteArray();  
 return Base64.encodeToString(byteArray, Base64.DEFAULT);  
}  
  
private String getRandomId() {  
 return String.valueOf((new Random()).nextLong());  
}

Для назначения изображениям идентификаторов используется класс Random. При переводе объекта Bitmap в строку происходит его псевдосжатие в формат PNG при помощи метода Bitmap.compress. Для перевода массива байтов в строку формата Base64 используется одноимённый класс.

**4.8** Обработка входящих GCM-сообщений

Для данной цели служат классы FirebaseMessagingService и PushHandler:

public class FirebaseMessagingService extends com.google.firebase.messaging.FirebaseMessagingService {  
  
 private static final String INTENT\_ACTION = "PUSH\_MESSAGE\_RECEIVED";  
  
 @Override  
 public void onMessageReceived(RemoteMessage remoteMessage) {  
 Intent intent = new Intent(INTENT\_ACTION);  
 intent.putExtra("id", remoteMessage.getData().get("id"));  
 LocalBroadcastManager.getInstance(this).sendBroadcast(intent);  
 }  
}

Класс FirebaseMessagingService при получении входящего сообщения вызывает метод LocalBroadcastManager.sendBroadcast, куда передаётся объект намерения, содержащий идентификатор изображения и строку-фильтр PUSH\_MESSAGE\_RECEIVED. Объект класса PushHandler, созданный активностью ColorizationActivity, принимает широковещательные сообщение именно с этим фильтром и затем вызывает запуск задачи DownloadingTask:

@Override  
public void onReceive(Context context, final Intent intent) {  
 activity.runOnUiThread(new Runnable() {  
 @Override  
 public void run() {  
 String id = intent.getStringExtra("id");  
 activity.currentTask = new DownloadingTask(id) {  
 @Override  
 protected void onPostExecute(Throwable tr) {  
 super.onPostExecute(tr);  
 activity.progressBar.setVisibility(View.GONE);  
 if (tr == null) {  
 activity.bitmap = getBitmap();  
 activity.imageView.setImageBitmap(activity.bitmap);  
 } else Toast.makeText(activity, tr.getMessage(),  
 Toast.LENGTH\_SHORT).show();  
 }  
 };  
 AsyncTaskCompat.executeParallel(activity.currentTask);  
 }  
 });  
}

При получении обработанного изображение оно отображается на экран при помощи метода ImageView.setImageBitmap. В случае любой ошибки, сообщение, которое она содержит, выводится на экран при помощи метода Toast.makeText.

**4.9** Модуль предоставляющий интерфейс для взаимодействия с сервером

В данном подразделе приведён код класса API, который в некоторой степени реализует паттерн Singleton.

public final class API {  
  
 private static final String SERVER\_ADDRESS = "http://192.168.0.104:8080/";  
  
 private static ServerInterface server;  
  
 private API() {  
  
 }  
  
 public static ServerInterface getInterface() {  
 if (server == null) server = new Retrofit.Builder()  
 .baseUrl(SERVER\_ADDRESS)  
 .addConverterFactory(ScalarsConverterFactory.create())  
 .build()  
 .create(ServerInterface.class);  
 return server;  
 }  
}

Данный класс содержит адрес сервера, заключённый в строку SERVER\_ADDRESS. Получение объекта, реализующего интерфейс ServerInterface происходит при помощи метода Retrofit.Builder.create, который является частью фреймворка Retrofit.

**4.10** Файлы AndroidManifest.xml и build.gradle

Файл манифеста используется при разработке Android-приложения, для описания основных компонентов приложения и их свойств, требующихся ресурсов, разрешения на использование которых будет получено у пользователя при установке приложения. Также данный файл содержит имя приложения и ссылки на «иконки» для приложения:

<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
 package="by.evgeniyshilov.colorizationclient">  
  
 <uses-permission android:name="android.permission.INTERNET" />  
 <uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" />  
  
 <application  
 android:allowBackup="true"  
 android:icon="@mipmap/ic\_launcher"  
 android:label="@string/app\_name"  
 android:roundIcon="@mipmap/ic\_launcher\_round"  
 android:supportsRtl="true"  
 android:theme="@style/AppTheme">  
  
 <service android:name=".gcm.FirebaseMessagingService">  
 <intent-filter>  
 <action android:name="com.google.firebase.MESSAGING\_EVENT" />  
 </intent-filter>  
 </service>  
  
 <activity android:name=".activities.PickerActivity">  
 <intent-filter>  
 <action android:name="android.intent.action.MAIN" />  
 <category android:name="android.intent.category.LAUNCHER" />  
 </intent-filter>  
 </activity>  
  
 <activity android:name=".activities.ColorizationActivity" />  
  
 </application>  
  
</manifest>

Файл build.gradle используется для указания информации о целевой и минимальной версии Android для разрабатываемого приложения, версии самого приложения. Также файл содержит зависимости, которые должна подгрузить Gradle, а также некоторые дополнительные параметры для системы сборки:

apply plugin: 'com.android.application'  
  
android {  
 compileSdkVersion 25  
 buildToolsVersion "25.0.3"  
 defaultConfig {  
 applicationId "by.evgeniyshilov.colorizationclient"  
 minSdkVersion 21  
 targetSdkVersion 25  
 versionCode 1  
 versionName "1.0"  
 testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"  
 }  
 buildTypes {  
 release {  
 minifyEnabled false  
 proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'  
 }  
 }  
}  
  
dependencies {  
 compile fileTree(dir: 'libs', include: ['\*.jar'])  
 androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {  
 exclude group: 'com.android.support', module: 'support-annotations'  
 })  
 compile 'com.android.support:appcompat-v7:25.3.1'  
 compile 'com.google.firebase:firebase-messaging:10.2.4'  
 compile 'com.squareup.retrofit2:retrofit:2.2.0'  
 compile 'com.squareup.retrofit2:converter-scalars:2.2.0'  
 testCompile 'junit:junit:4.12'  
}  
  
apply plugin: 'com.google.gms.google-services'

Версии указываются внутри структуры defaultConfig. Дополнительные библиотеки указаны внутри структуры dependencies.