

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики. Кафедра програмного забезпечення
комп'ютерних систем

ЗВІТ
з лабораторної роботи № 5
«ОБРОБКА ДАНИХ У ЗАСТОСУНКАХ ANDROID»

Виконав:

студент 3-го курсу, групи КП-83,
спеціальності 121 – Інженерія
програмного забезпечення
Коваль Андрій Олександрович

Перевірив:

к. т. н, старший викладач
Хайдуров Владислав
Володимирович

Київ – 2020

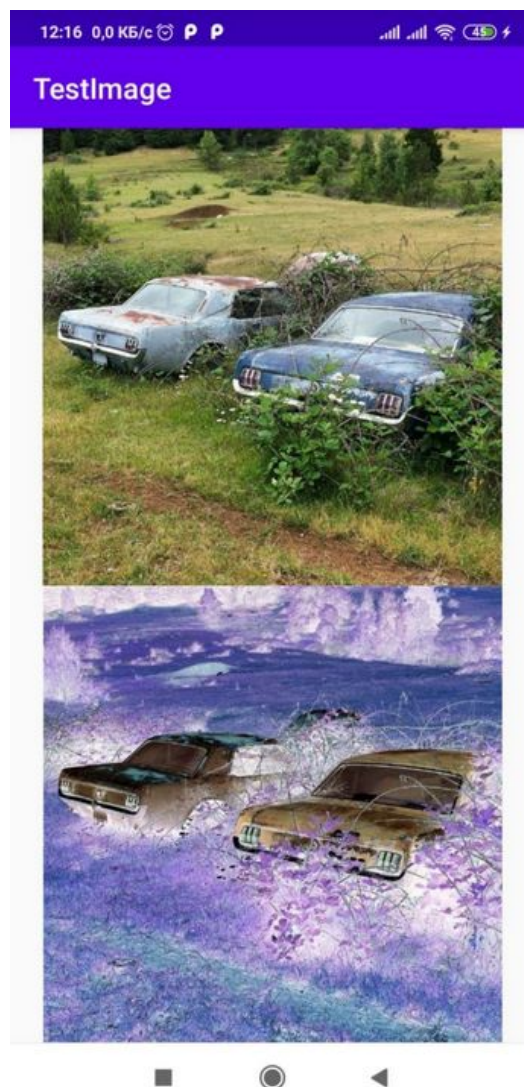
ВСТУП	2
Завдання до лабораторної роботи	3
Короткі теоретичні відомості	8
Програмна реалізація задачі	9
Завдання 6	29
Висновки	32

ВСТУП

Метою даної роботи є ознайомлення із основними принципами обробки графічних даних з використанням основних методів і алгоритмів цифрової обробки зображень.

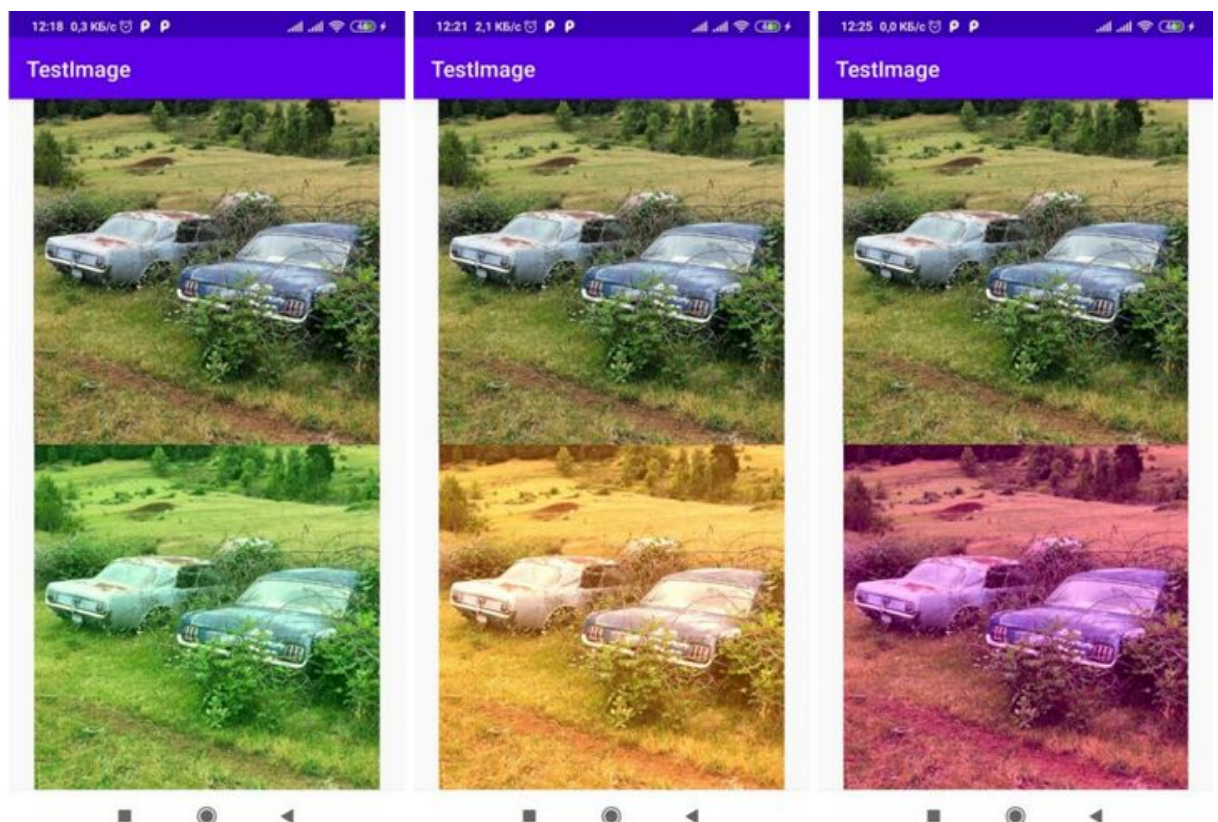
Завдання до лабораторної роботи

1. Ознайомитись із усіма теоретичними відомостями до лабораторної роботи.
2. Створити мобільний застосунок, який виконує інвертування кольорової палітри попередньо підготовленого зображення, що міститься у папці drawable проекту програмного застосунку. Вивести вхідне та отримане зображення в об'єкти типу ImageView.

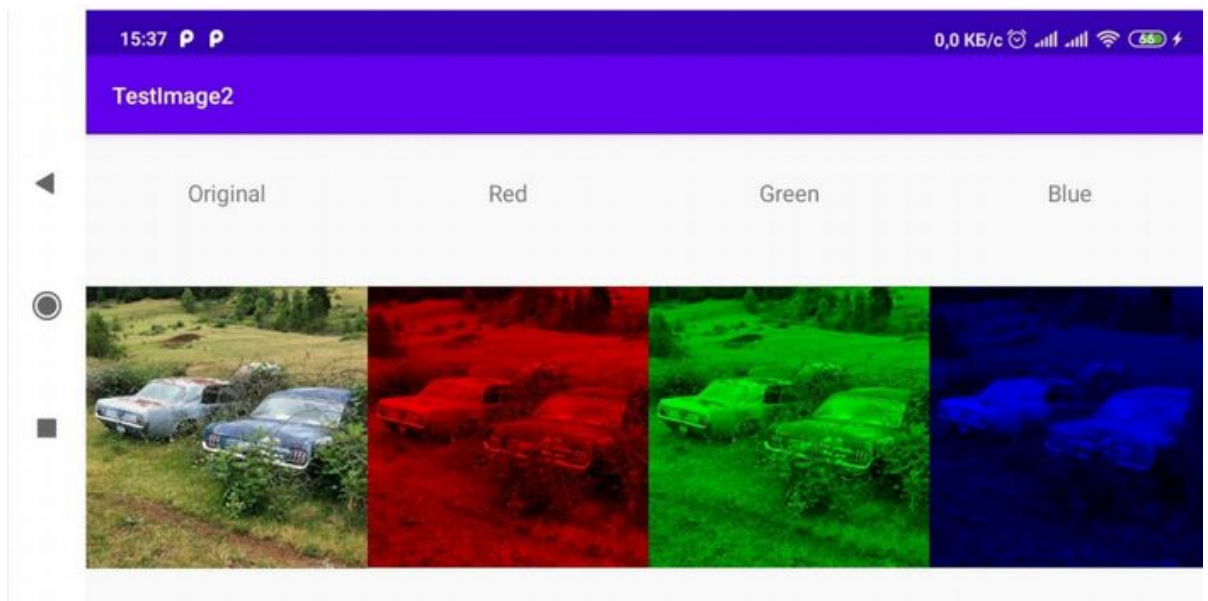




3. Створити мобільний застосунок, який для деякої (однієї з трьох) компонент виконує зміну, наприклад, до кожного пікселя додає якесь постійне значення. Вивести усі зображення в об'єкти типу `ImageView`.



4. Створити мобільний застосунок, за допомогою якого виконується розбивка зображення на компоненти Red, Green та Blue.

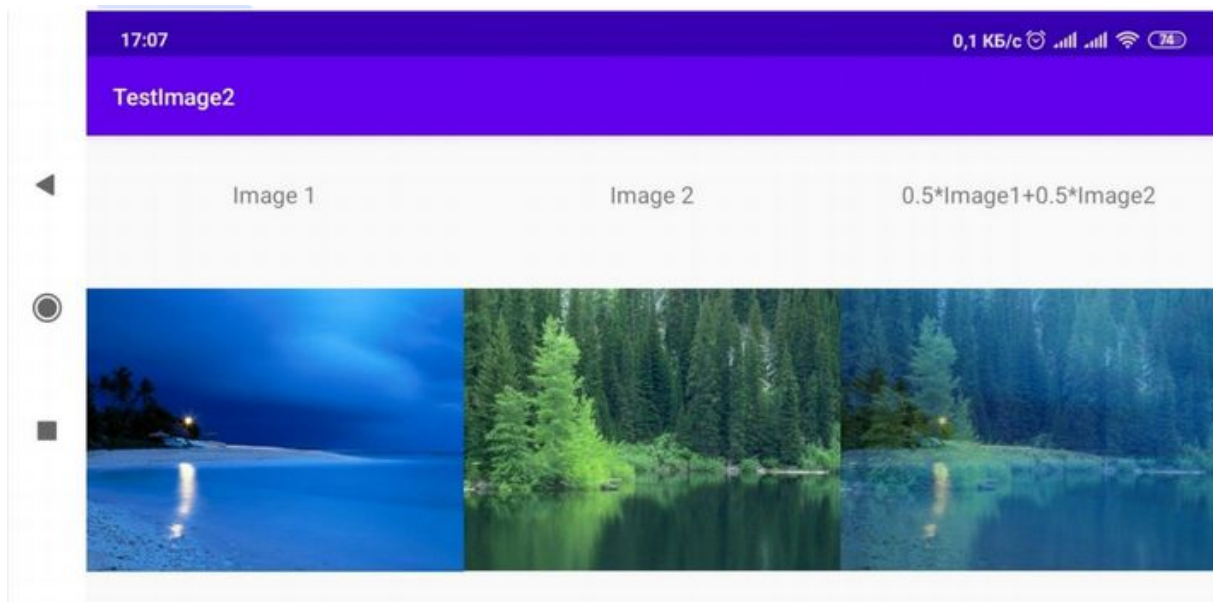


5. Створити мобільний застосунок, за допомогою якого виконується «злиття» двох зображень у певних пропорціях (долях від одиниці, яка береться за 100%). Наприклад, якщо «вклад» першого зображення у нове складає 0.4, то «вклад» другого – $1 - 0.4 = 0.6$. У загальному «вклад» першого зображення позначимо через α , а «вклад» другого – $(1 - \alpha)$. Тоді шукане зображення можна записати у наступному вигляді:

$$Image_{new} = \alpha \cdot Image_1 + (1 - \alpha)Image_2.$$

Вхідні файли також попередньо підготувати та додати їх до папки

drawable.



За додаткові бали передбачити те, що α – змінна величина, $\alpha \in (0; 1)$. Крок зміни обрати самостійно. Вивести анімацію зображень на екран мобільного застосунку.

6. Створити мобільний застосунок, що виконує фільтрацію вхідного зображення з використанням матричних фільтрів, що описані у лабораторній роботі (фільтр розмиття, фільтр поліпшення чіткості, медіанний фільтр, фільтр ерозії і нарощування та фільтр Собеля).
7. Створити мобільний застосунок, що виконує вбудовування водяного знаку зображення, що попередньо підготовлені та містяться в папці drawable мобільного застосунку. Для процесу вбудовування водяний знак перетворити до бінарного (чорно-білого, не сірого!). Використати для вбудовування метод найменшого значущого біта. Параметр «номер бітової площини» (натуральне число, менше за 9) вивести на форму (Activity) для відображення різниці в результатах вбудовування. Водяний знак вбудовувати у канал Blue. Передбачити випадок різних розмірів вхідного зображення (контейнера) та водяного знаку. Якщо вхідне зображення більше за водяний знак, виконати вбудовування циклічно та періодично для всього вхідного

зображення (замостити вхідне зображення водяним знаком). Якщо вхідне зображення менше за водяний знак, вбудувати частину водяного знаку для всього вхідного зображення.

8. Додатковий бал передбачається з встворення методу, який виконує вилучення водяного знаку із заповненого контейнера (зображення з водяним знаком) для завдання 7 даної лабораторної роботи.
9. Усі завдання лабораторної роботи за бажанням можна виконати в одному багатовіконному мобільному додатку.

Короткі теоретичні відомості

Під час роботи із зображеннями ми будемо використовувати найбільш поширену на даний момент конфігурацію формування пікселів ARGB_8888. Перший октет займає альфа канал, а усі наступні червоний R, зелений G та синій B по черзі. В програмній реалізації ми керуємо масивом типу `int`, кожен елемент якого має якраз 32 біти, які і потрібні для утримання у собі інформації про пікселі.

Усі завдання лабораторної роботи виконуються, використовуючи пікселі напряму, що допоможе суттєво розібратися у процесах формування та обробки зображень

Програмна реалізація задачі

Було зроблено один додаток з різними Activity, кожна з яких представляє окреме завдання. Код переходу до активностей та головну розмітку опустимо, оскільки це не є основним завданням роботи та не представляло ніяких складностей.

Для маніпуляцій з кольорами було написано власний клас `MyColor`, щоб не використовувати вбудований клас `Color` й більш глибоко зрозуміти внутрішню будову взаємодію з кольорами. Також для спрощення роботи було створення перечислення `MyColorEnum` та клас `ImageUtils`.

MyColor

```
package com.example.lab5;

public class MyColor {

    private int _r, _g, _b;
    private int _alpha;

    public MyColor(int r, int g, int b) {
        this._r = trimToBounds(r);
        this._g = trimToBounds(g);
        this._b = trimToBounds(b);
        this._alpha = 255;
    }

    public MyColor(int r, int g, int b, int alpha) {
        this(r, g, b);
        this._alpha = trimToBounds(alpha);
    }

    public MyColor(int srcColor) {
        this((((srcColor & 0xff0000) >> 16) & 0x000000ff),
            (((srcColor & 0x00ff00) >> 8) & 0x000000ff),
            (srcColor & 0x000000ff),
            ((srcColor & 0xff000000) >> 24) & 0x000000ff);
    }

    public MyColor(MyColor src) {
        this(src.getR(), src.getG(), src.getB(), src.getAlpha());
    }

    public int toInt() {
        return (_alpha << 24) + (_r << 16) + (_g << 8) + _b;
    }
}
```

```

    public int getAlpha() {
        return _alpha;
    }

    public int getR() {
        return _r;
    }

    public int getG() {
        return _g;
    }

    public int getB() {
        return _b;
    }

    public void setAlpha(int alpha) {
        this._alpha = trimToBounds(alpha);
    }

    public void setR(int r) {
        this._r = trimToBounds(r);
    }

    public void setG(int g) {
        this._g = trimToBounds(g);
    }

    public void setB(int b) {
        this._b = trimToBounds(b);
    }

    private int trimToBounds(int value) {
        if (value > 255) {
            value = 255;
        } else if (value < 0) {
            value = 0;
        }
        return value;
    }
}

```

MyColorEnum

```

public enum MyColorEnum {
    RED, GREEN, BLUE
}

```

ImageUtils

```

public class ImageUtils {

    public static void setImagePixels(int[] pixels, ImageView dstImageView,
int width, int height) {
        Bitmap dstBitmap = Bitmap.createBitmap(width, height,
Bitmap.Config.ARGB_8888);

```

```
dstBitmap.setPixels(pixels, 0, width, 0, 0, width, height);  
dstImageView.setImageBitmap(dstBitmap);  
}
```

```
}
```

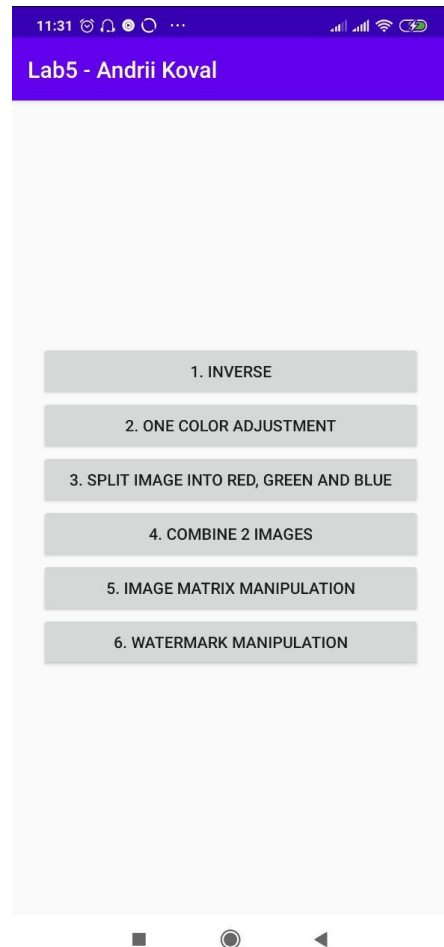


Рис. 1, зовнішній вигляд головного меню

Завдання 1

Інвертування кольорів є досить простою задачею - просто віднімаємо від 255 (максимальне значення одного біта кольору) саме значення та отримуємо інвертований колір.

InverseImageActivity

```
public class InverseImageActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_inverse_image);
        this.invertSecondImage();
    }

    private void invertSecondImage() {
        Bitmap src = BitmapFactory.decodeResource(getResources(),
R.mipmap.tesla);
        int width = src.getWidth();
        int height = src.getHeight();
        int[] srcPixels = new int[width * height];
        src.getPixels(srcPixels, 0, width, 0, 0, width, height);
        int[] dstPixels = new int[width * height];
        this.invertPixels(srcPixels, dstPixels);

        ImageView dstImageView = findViewById(R.id.dst);
        Bitmap dstBitmap = Bitmap.createBitmap(width, height,
Bitmap.Config.ARGB_8888);
        dstBitmap.setPixels(dstPixels, 0, width, 0, 0, width, height);
        dstImageView.setImageBitmap(dstBitmap);
        src.recycle();
    }

    private void invertPixels(int[] src, int[] dst) {
        for (int i = 0; i < src.length; i++) {
            MyColor srcColor = new MyColor(src[i]);
            int newR = 255 - srcColor.getR();
            int newG = 255 - srcColor.getG();
            int newB = 255 - srcColor.getB();
            dst[i] = new MyColor(newR, newG, newB).toInt();
        }
    }
}
```

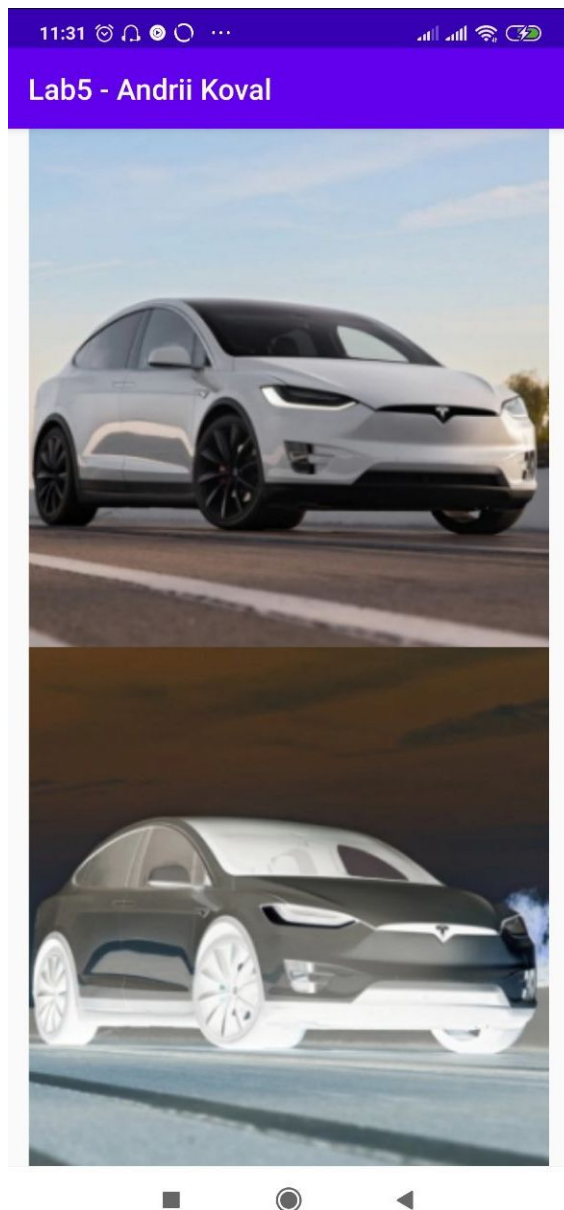


Рис 2-1, Інвертування зображення

Завдання 2

У другому завданні ми змінюємо один канал кольору на якесь значення. У даній реалізації є можливість вибору кольору, яких ми змінюємо.

OneColorAdjustmentActivity

```
public class OneColorAdjustmentActivity extends AppCompatActivity {

    private int COLOR_ADJUSTMENT_VALUE = 50;
    private MyColorEnum DEFAULT_CHECKED_COLOR = MyColorEnum.GREEN;

    View.OnClickListener radioButtonClickListener = new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            RadioButton rb = (RadioButton)v;
            switch (rb.getId()) {
                case R.id.radio_red: adjustOneColor(MyColorEnum.RED,
COLOR_ADJUSTMENT_VALUE);
                    break;
                case R.id.radio_green: adjustOneColor(MyColorEnum.GREEN,
COLOR_ADJUSTMENT_VALUE);
                    break;
                case R.id.radio_blue: adjustOneColor(MyColorEnum.BLUE,
COLOR_ADJUSTMENT_VALUE);
                    break;
            }
        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_one_color_adjustment);
        this.setUpRadioButtons();
        this.adjustOneColor(DEFAULT_CHECKED_COLOR, COLOR_ADJUSTMENT_VALUE);
    }

    // ...

    private void adjustOneColor(MyColorEnum color, int value) {
        Bitmap src = BitmapFactory.decodeResource(getResources(),
R.mipmap.dream_car_foreground);
        int width = src.getWidth();
        int height = src.getHeight();
        int[] srcPixels = new int[width * height];
        src.getPixels(srcPixels, 0, width, 0, 0, width, height);
        int[] dstPixels = new int[width * height];
        adjustPixels(srcPixels, dstPixels, color, value);

        ImageView dstImageView = findViewById(R.id.dst);
        Bitmap dstBitmap = Bitmap.createBitmap(width, height,
Bitmap.Config.ARGB_8888);
```

```

        dstBitmap.setPixels(dstPixels, 0, width, 0, 0, width, height);
        dstImageView.setImageBitmap(dstBitmap);
        src.recycle();
    }

    private void adjustPixels(int[] src, int[] dst, MyColorEnum color, int
value) {
        for (int i = 0; i < src.length; i++) {
            MyColor srcColor = new MyColor(src[i]);
            switch (color) {
                case RED: srcColor.setR(srcColor.getR() + value); break;
                case GREEN: srcColor.setG(srcColor.getG() + value); break;
                case BLUE: srcColor.setB(srcColor.getB() + value); break;
            }
            dst[i] = srcColor.toInt();
        }
    }
}

```

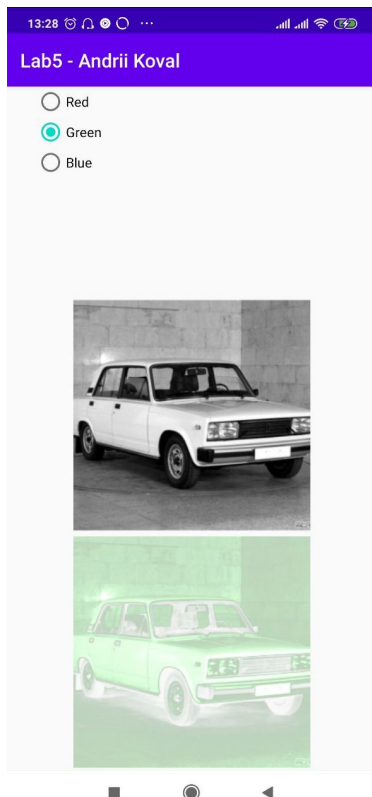


Рис 3-1, додавання
зеленого кольору

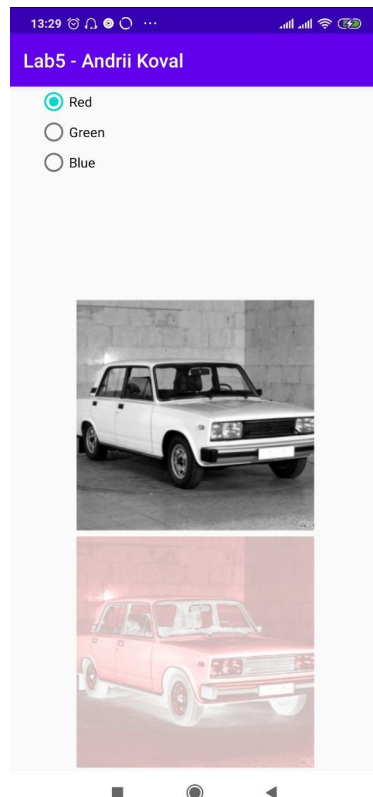


Рис 3-2, додавання
червоного кольору

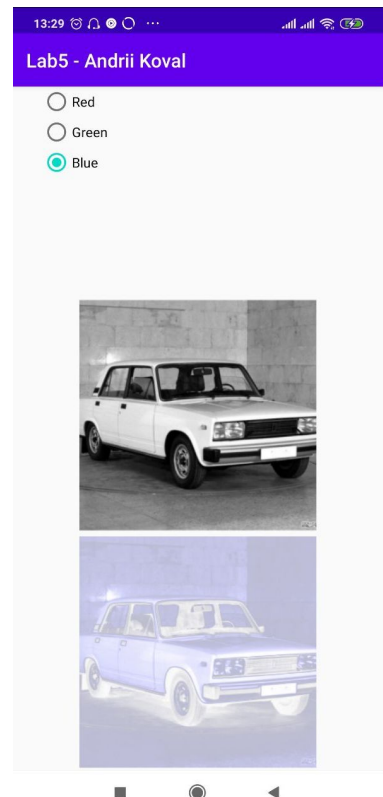


Рис 3-3, додавання
синього кольору

Завдання 3

У третьому завданні було розділено зображення на 3 кольорові канали. У кожному каналі присутній лише 1 колір - усі інші 0, окрім каналу прозорості.

SplitIntoColorsActivity

```
public class SplitIntoColorsActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_split_into_colors);
        this.createThreeImagesAndFillThem();
    }

    private void createThreeImagesAndFillThem() {
        Bitmap src = BitmapFactory.decodeResource(getResources(),
R.mipmap.android_foreground);
        int width = src.getWidth();
        int height = src.getHeight();
        int[] srcPixels = new int[width * height];
        int[] bufferPixels = new int[srcPixels.length];
        src.getPixels(srcPixels, 0, width, 0, 0, width, height);

        getRedPicture(srcPixels, bufferPixels);
        setImagePixels(bufferPixels, R.id.redImage, width, height);

        getGreenPicture(srcPixels, bufferPixels);
        setImagePixels(bufferPixels, R.id.greenImage, width, height);

        getBluePicture(srcPixels, bufferPixels);
        setImagePixels(bufferPixels, R.id.blueImage, width, height);
        src.recycle();
    }

    private void getRedPicture(int[] src, int[] dst) {
        for (int i = 0; i < src.length; i++) {
            MyColor color = new MyColor(src[i]);
            color.setG(0);
            color.setB(0);
            dst[i] = color.toInt();
        }
    }

    private void getGreenPicture(int[] src, int[] dst) {
        for (int i = 0; i < src.length; i++) {
            MyColor color = new MyColor(src[i]);
            color.setR(0);
            color.setB(0);
            dst[i] = color.toInt();
        }
    }
}
```

```
    }  
    }  
  
    private void getBluePicture(int[] src, int[] dst) {  
        for (int i = 0; i < src.length; i++) {  
            MyColor color = new MyColor(src[i]);  
            color.setR(0);  
            color.setG(0);  
            dst[i] = color.toInt();  
        }  
    }  
  
    private void setImagePixels(int[] pixels, @IdRes int resId, int width,  
int height) {  
        ImageUtils.setImagePixels(pixels, (ImageView) findViewById(resId), width,  
height);  
    }  
}
```

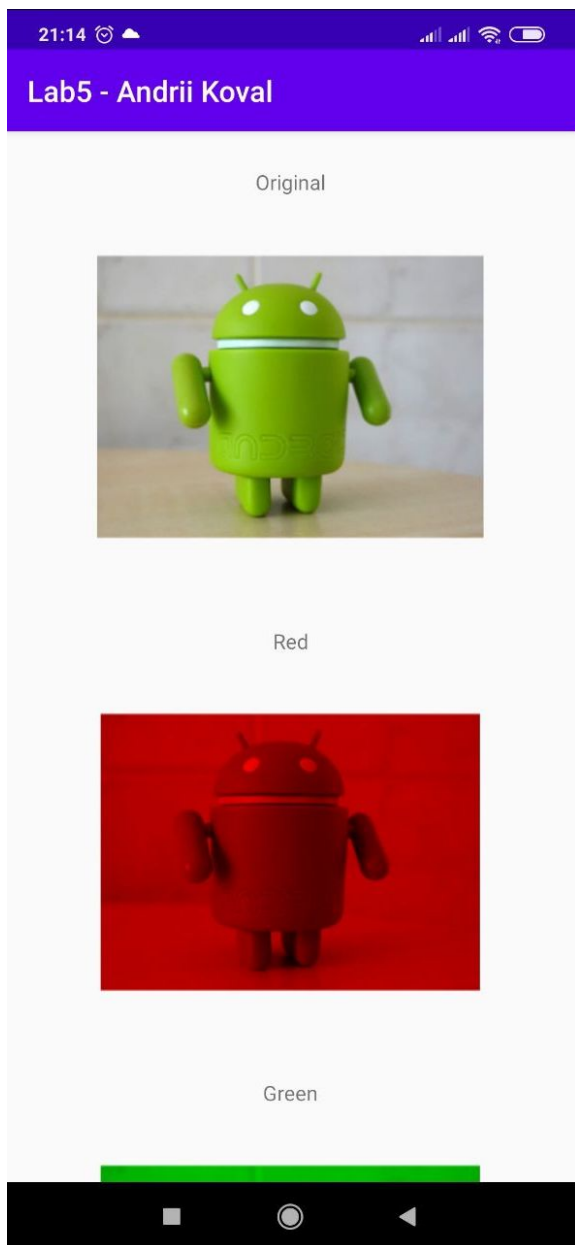


Рис 4-1. Оригінал та червоний канали

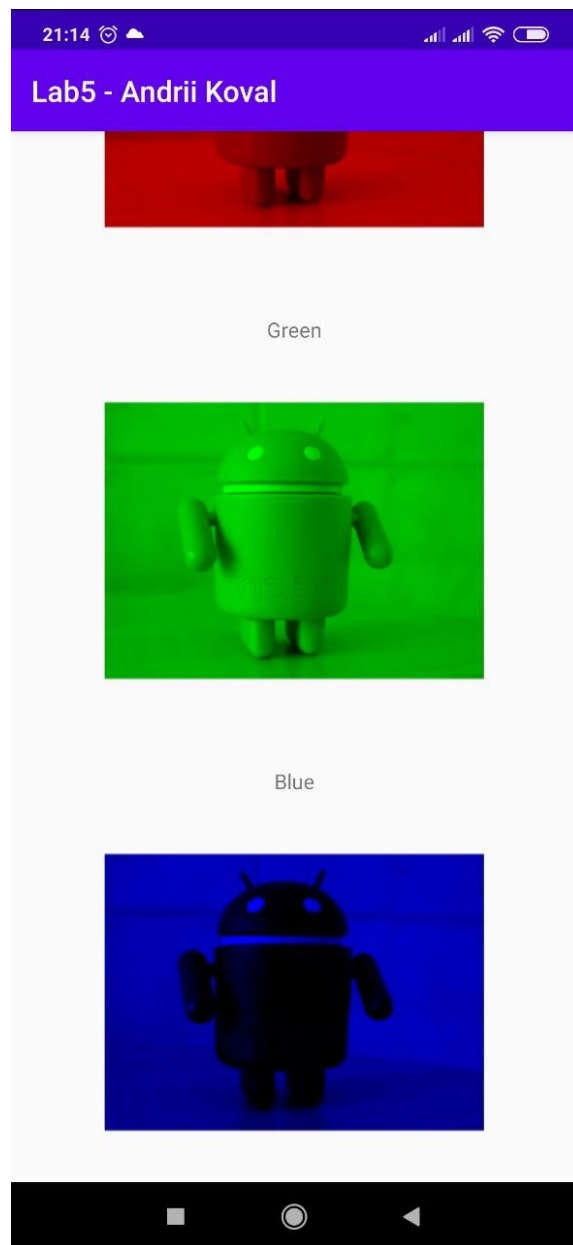


Рис 4-2. Зелений та синій канали

Завдання 4

У цьому завданні ми поєднуємо 2 картинки, 2 світи, 2 епохи автопрому за допомогою нескладної математики

$$Image_{new} = \alpha \cdot Image_1 + (1 - \alpha)Image_2.$$

CombineImagesActivity

```
public class CombineImagesActivity extends AppCompatActivity {

    private Bitmap imageBitmap1, imageBitmap2;
    private TextView alphaTextView;
    private int width, height;
    private int[] pixels1, pixels2;

    private ImageView imageView;

    private float INITIAL_ALPHA = 0.3f;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_combine_images);
        this.initImages();
        this.initTextView();
        this.setSeekBarEventListener();
        this.onAlphaChange(INITIAL_ALPHA);
    }

    // ...

    private void setSeekBarEventListener() {
        SeekBar seekBar = findViewById(R.id.seekBar);
        seekBar.setProgress((int)(INITIAL_ALPHA * 100));
        if (seekBar != null) {
            seekBar.setOnSeekBarChangeListener(new
SeekBar.OnSeekBarChangeListener() {
                @Override
                public void onProgressChanged(SeekBar seekBar, int
progress, boolean fromUser) {
                    onAlphaChange(progress / 100.0f);
                }

                @Override
                public void onStartTrackingTouch(SeekBar seekBar) { }

                @Override
                public void onStopTrackingTouch(SeekBar seekBar) { }
            });
        }

    private void onAlphaChange(float alpha) {
        alphaTextView.setText(Float.toString(alpha));
    }
}
```

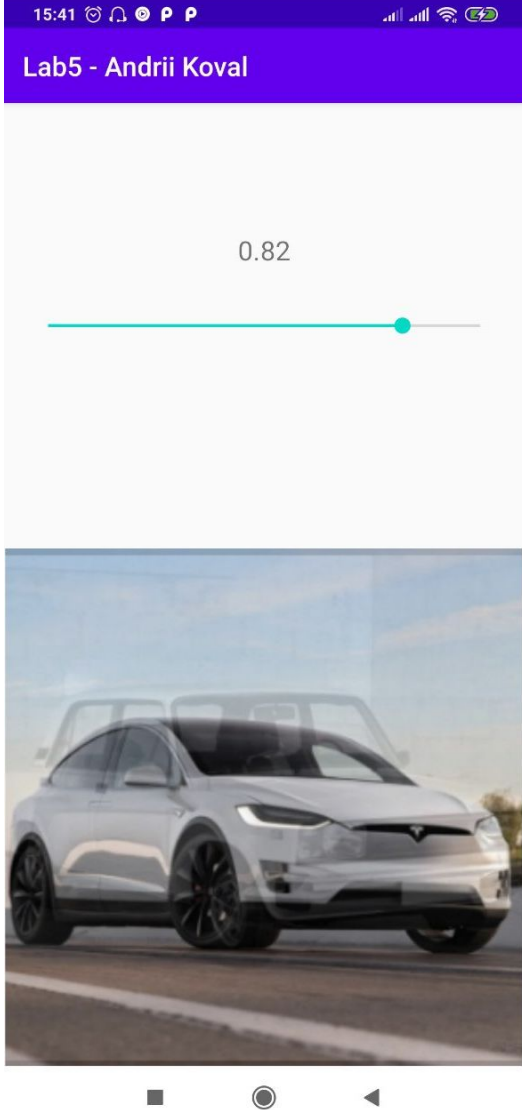
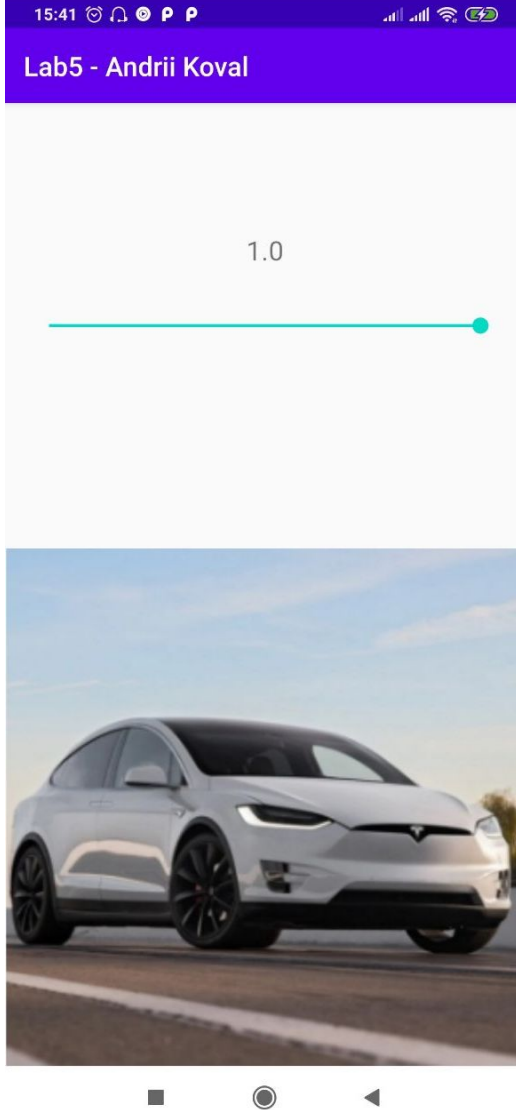


```
        combineImages(alpha);
    }

    private void combineImages(float alpha) {
        int[] dst = new int[pixels1.length];
        for (int i = 0; i < pixels1.length; i++) {
            MyColor color1 = new MyColor(pixels1[i]);
            MyColor color2 = new MyColor(pixels2[i]);
            int r = (int)(color1.getR() * alpha + (1 - alpha) *
color2.getR());
            int g = (int)(color1.getG() * alpha + (1 - alpha) *
color2.getG());
            int b = (int)(color1.getB() * alpha + (1 - alpha) *
color2.getB());
            dst[i] = new MyColor(r, g, b).toInt();
        }

        ImageUtils.setImagePixels(dst, imageView, width, height);
    }
}
```

<div data-bbox="304 203 826 320"> <div>15:41</div> <div> </div> <div>Lab5 - Andrii Koval</div> </div> <div data-bbox="304 320 826 757"> <div>0.0</div> <div> </div> </div> <div data-bbox="304 757 826 1279"> </div> <div data-bbox="304 1279 826 1361"> <div> </div> </div>	<div data-bbox="858 203 1380 320"> <div>15:41</div> <div> </div> <div>Lab5 - Andrii Koval</div> </div> <div data-bbox="858 320 1380 757"> <div>0.41</div> <div> </div> </div> <div data-bbox="858 757 1380 1279"> </div> <div data-bbox="858 1279 1380 1361"> <div> </div> </div>
<div>Рис. 5-1. Перше зображення без домішок другого</div>	<div>Рис. 5-2. Початок процесу об'єднання</div>

 <p>The screenshot shows a mobile application interface with a purple header bar containing the text "Lab5 - Andrii Koval". Below the header, a slider control is displayed with the value "0.82". The slider has a teal track and a teal knob. Below the slider is a photograph of a silver Tesla Model X. A semi-transparent, ghosted image of the same car is overlaid on top of the main image, slightly offset to the left and back, illustrating the "fusion" process mentioned in the caption.</p>	 <p>The screenshot shows the same mobile application interface as Figure 5-3. The slider control now displays the value "1.0", and the teal knob is positioned at the right end of the track. The photograph of the silver Tesla Model X is shown without the ghost overlay, representing the "second image without the influence of the first" as described in the caption.</p>
<p>Рис. 5-3. Закінчення процесу об'єднання</p>	<p>Рис. 5-4. Друге зображення без домішок першого</p>

Завдання 5

У даному завданні ми використовуємо матриці згортки для різних маніпуляцій із зображенням, такими як розмиття, збільшення різкості, зменшення різкості, знаходження градієнтних точок переходу кольорів та висвітлення. Було створено окремий клас `MatrixManipulator`, який ми використовуємо як колбек для обробки одного пікселя, коли ми маємо матрицю згортки та матрицю пікселів.

MatrixManipulator
<pre>public abstract class MatrixManipulator { public abstract int getColor(double[][] matrix, int[][] pixels); }</pre>
Приклад використання
<pre>private Bitmap applySharpnessMatrix(Bitmap bitmap) { double[][] matrix = new double[][] { new double[] { -1, -1, -1 }, new double[] { -1, 9, -1 }, new double[] { -1, -1, -1 }, }; return applyMatrix(matrix, bitmap, defaultMatrixManipulator, true); } ... private MatrixManipulator defaultMatrixManipulator = new MatrixManipulator() { @Override public int getColor(double[][] matrix, int[][] pixels) { int size = matrix.length; int sumR = 0, sumG = 0, sumB = 0; for(int i = 0; i < size; ++i) { for(int j = 0; j < size; ++j) { MyColor color = new MyColor(pixels[i][j]); sumR += (color.getR() * matrix[i][j]); sumG += (color.getG() * matrix[i][j]); sumB += (color.getB() * matrix[i][j]); } } int center = (int)Math.floor(size / 2); int alpha = new MyColor(pixels[center][center]).getAlpha(); MyColor color = new MyColor((int)sumR, (int)sumG, (int)sumB, alpha); return color.toInt(); } }</pre>

```
    }  
};
```

Прикладається повний код активності з матрицями для більш детального ознайомлення

MatrixManipulationActivity

```
public class MatrixManipulationActivity extends AppCompatActivity {  
    private ImageView imageViewOriginal, imageViewBlur, imageViewSharpness,  
        imageViewMedian, imageViewErosionAndGrowth,  
    imageViewSobel;  
  
    private Bitmap srcBitmap;  
  
    private int MEDIAN_MATRIX_SIZE = 10;  
    private int SRC_IMAGE_RES_ID = R.mipmap.tesla;  
  
    private MatrixManipulator defaultMatrixManipulator = new MatrixManipulator()  
{  
    @Override  
    public int getColor(double[][] matrix, int[][] pixels) {  
        int size = matrix.length;  
        int sumR = 0, sumG = 0, sumB = 0;  
  
        for(int i = 0; i < size; ++i) {  
            for(int j = 0; j < size; ++j) {  
                MyColor color = new MyColor(pixels[i][j]);  
                sumR += (color.getR() * matrix[i][j]);  
                sumG += (color.getG() * matrix[i][j]);  
                sumB += (color.getB() * matrix[i][j]);  
            }  
        }  
  
        int center = (int)Math.floor(size / 2);  
        int alpha = new MyColor(pixels[center][center]).getAlpha();  
        MyColor color = new MyColor((int)sumR, (int)sumG, (int)sumB, alpha);  
        return color.toInt();  
    }  
};  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_matrix_manipulation);  
        initImageViews();  
        initSrcBitmap();  
        applyMatrices();  
    }  
}
```

```

private void initImageViews() {
    this.imageViewOriginal = findViewById(R.id.src);
    this.imageViewBlur = findViewById(R.id.imageBlur);
    this.imageViewSharpness = findViewById(R.id.imageSharpness);
    this.imageViewErosionAndGrowth =
findViewById(R.id.imageErosionAndGrowth);
    this.imageViewMedian = findViewById(R.id.imageMedian);
    this.imageViewSobel = findViewById(R.id.imageSobel);
}

private void initSrcBitmap() {
    this.srcBitmap = BitmapFactory.decodeResource(getResources(),
SRC_IMAGE_RES_ID);
    this.imageViewOriginal.setImageBitmap(srcBitmap);
}

private Bitmap applyMatrix(double[][] matrix, Bitmap src, MatrixManipulator
matrixManipulator, boolean horizontal) {
    int width = src.getWidth();
    int height = src.getHeight();

    int size = matrix.length;

    Bitmap result = Bitmap.createBitmap(width, height, src.getConfig());

    int[][] pixels = new int[size][size];

    if (horizontal) {
        for(int y = 0; y < height - size; y++) {
            for(int x = 0; x < width - size; x++) {

                for(int i = 0; i < size; i++) {
                    for(int j = 0; j < size; j++) {
                        pixels[i][j] = src.getPixel(x + i, y + j);
                    }
                }

                result.setPixel(x + 1, y + 1,
matrixManipulator.getColor(matrix, pixels));
            }
        }
    } else {
        for(int x = 0; x < width - size; x++) {
            for(int y = 0; y < height - size; y++) {

                for(int i = 0; i < size; i++) {
                    for(int j = 0; j < size; j++) {
                        pixels[i][j] = src.getPixel(x + i, y + j);
                    }
                }

                result.setPixel(x + 1, y + 1,
matrixManipulator.getColor(matrix, pixels));
            }
        }
    }
}

```



```

        return result;
    }

    private Bitmap applyBlurMatrix(Bitmap bitmap) {
        double[][] matrix = new double[][] {
            new double[] { .000789, .006581, .013347, .000789, .006581 },
            new double[] { .006581, .054901, .111345, .054901, .006581 },
            new double[] { .013347, .111345, .225821, .111345, .013347 },
            new double[] { .006581, .054901, .111345, .054901, .006581 },
            new double[] { .000789, .006581, .013347, .000789, .006581 }
        };

        return applyMatrix(matrix, bitmap, defaultMatrixManipulator, true);
    }

    private Bitmap applyMedianMatrix(Bitmap bitmap, int size) {
        return applyMatrix(new double[size][size], bitmap, new
MatrixManipulator() {
            @Override
            public int getColor(double[][] matrix, int[][] pixels) {
                int[] oneDimensionPixels = new int[pixels.length *
pixels.length];
                int index = 0;
                for (int[] pixelsRow : pixels) {
                    for (int pixel : pixelsRow) {
                        oneDimensionPixels[index++] = pixel;
                    }
                }
                // to clear memory;
                pixels = null;
                //
                Arrays.sort(oneDimensionPixels);
                int leftCenterIndex = (int)Math.floor(oneDimensionPixels.length
/ 2.0) - 1;
                if (oneDimensionPixels.length % 2 == 0) {
                    MyColor color1 = new
MyColor(oneDimensionPixels[leftCenterIndex]);
                    MyColor color2 = new
MyColor(oneDimensionPixels[leftCenterIndex + 1]);
                    return new MyColor((color1.getR() + color2.getR()) / 2,
                        (color1.getG() + color2.getG()) / 2,
                        (color1.getB() + color2.getB()) / 2,
                        (color1.getAlpha() + color2.getAlpha()) /
2).toInt());
                }
                return oneDimensionPixels[leftCenterIndex + 1];
            }
        }, true);
    }

    private Bitmap applySharpnessMatrix(Bitmap bitmap) {
        double[][] matrix = new double[][] {
            new double[] { -1, -1, -1 },
            new double[] { -1, 9, -1 },
            new double[] { -1, -1, -1 },
        };

        return applyMatrix(matrix, bitmap, defaultMatrixManipulator, true);
    }

```

```

    }

    private Bitmap applyErosionAndGrowthMatrix(Bitmap bitmap) {
        double[][] matrix = new double[][] {
            new double[] { .0, .0, .1, .0, .0 },
            new double[] { .0, .1, .1, .1, .0 },
            new double[] { .1, .1, .1, .1, .1 },
            new double[] { .0, .1, .1, .1, .0 },
            new double[] { .0, .0, .1, .0, .0 },
        };

        return applyMatrix(matrix, bitmap, defaultMatrixManipulator, true);
    }

    private Bitmap applySobelMatrix(Bitmap bitmap) {
        double[][] horizontalMatrix = new double[][] {
            new double[] { -1, -2, -1 },
            new double[] { 0, 0, 0 },
            new double[] { 1, 2, 1 },
        };

        double[][] verticalMatrix = new double[][] {
            new double[] { -1, 0, 1 },
            new double[] { -2, 0, 2 },
            new double[] { -1, 0, 1 },
        };

        Bitmap horizontallyDone = applyMatrix(horizontalMatrix, bitmap,
        defaultMatrixManipulator, true);
        Bitmap verticallyDone = applyMatrix(verticalMatrix, bitmap,
        defaultMatrixManipulator, false);
        int width = horizontallyDone.getWidth();
        int height = horizontallyDone.getHeight();
        Bitmap finalResult = Bitmap.createBitmap(width, height,
        Bitmap.Config.ARGB_8888);
        // will use this as a buffer for memory economy
        int[] horizontalPixels = new int[width * width];
        //
        int[] verticalPixels = new int[horizontalPixels.length];

        horizontallyDone.getPixels(horizontalPixels, 0, width, 0, 0, width,
height);
        verticallyDone.getPixels(verticalPixels, 0, width, 0, 0, width, height);

        // copied from prev task
        float alpha = .5f;
        //

        for (int i = 0; i < horizontalPixels.length; i++) {
            MyColor color1 = new MyColor(horizontalPixels[i]);
            MyColor color2 = new MyColor(verticalPixels[i]);
            int r = (int)(color1.getR() * alpha + (1 - alpha) * color2.getR());
            int g = (int)(color1.getG() * alpha + (1 - alpha) * color2.getG());
            int b = (int)(color1.getB() * alpha + (1 - alpha) * color2.getB());
            horizontalPixels[i] = new MyColor(r, g, b).toInt();
        }
    }

```

```
        finalResult.setPixels(horizontalPixels, 0, width, 0, 0, width, height);
        return finalResult;
    }

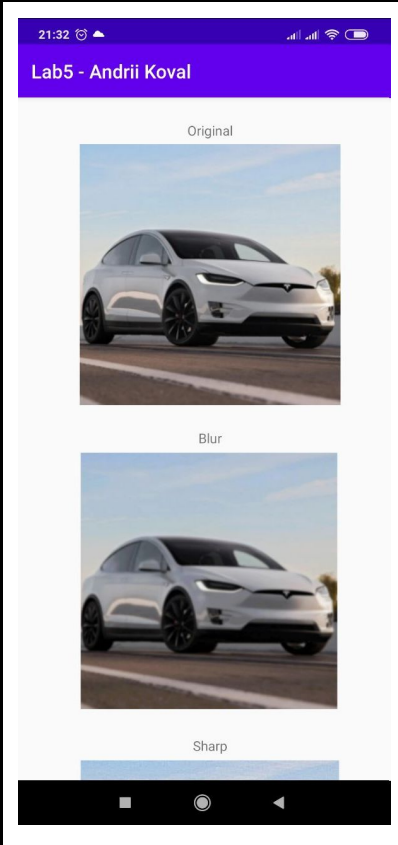
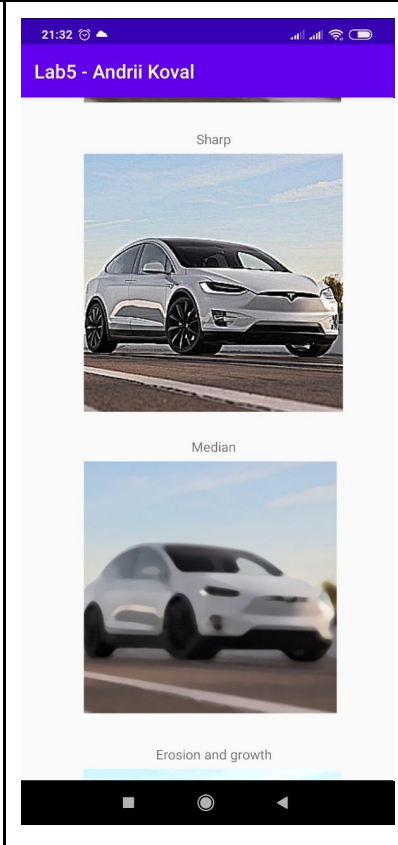
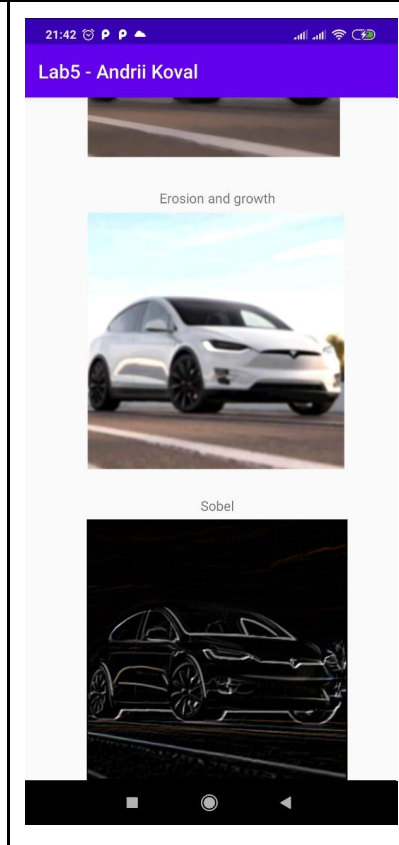
    private void applyMatrices() {
        Bitmap blurred = applyBlurMatrix(srcBitmap);
        this.imageViewBlur.setImageBitmap(blurred);

        Bitmap sharpened = applySharpnessMatrix(srcBitmap);
        this.imageViewSharpness.setImageBitmap(sharpened);

        Bitmap medianned = applyMedianMatrix(srcBitmap, MEDIAN_MATRIX_SIZE);
        this.imageViewMedian.setImageBitmap(medianned);

        Bitmap erosionAndGrowthBitmap = applyErosionAndGrowthMatrix(srcBitmap);
        this.imageViewErosionAndGrowth.setImageBitmap(erosionAndGrowthBitmap);

        Bitmap sobelBitmap = applySobelMatrix(srcBitmap);
        this.imageViewSobel.setImageBitmap(sobelBitmap);
    }
}
```

		
Рис 6-1, оригінал та фільтр розмиття	Рис 6-2, фільтр різкості та медіанний	Рис 6-2, фільтр ерозії та росту та Собеля

Завдання 6

Я трохи змінив завдання та зробив кодування секретного рядку в картинку. Рядок кодувався за випадковим ключем, за допомогою якого його можна потім потенційно витягнути звідти

Кодування рядку в зображення

```
private Bitmap makeWatermarkedImage(Bitmap src, String secret, String key) {
    int width = src.getWidth();
    int height = src.getHeight();
    Bitmap watermarked = Bitmap.createBitmap(width, height,
    Bitmap.Config.ARGB_8888);
    int[] watermarkedPixels = new int[width * height];
    src.getPixels(watermarkedPixels, 0, width, 0, 0, width, height);
    String secretBinary = stringToBinary(secret);
    String keyBinary = stringToBinary(key);

    for (int i = 0; i < keyBinary.length(); i++) {
```

```

int secretBit = Integer.parseInt(String.valueOf(secretBinary.charAt(i)));
if (keyBinary.charAt(i) == '0') {
    if (watermarkedPixels[i] % 2 == 0) {
        watermarkedPixels[i] |= secretBit;
    } else {
        watermarkedPixels[i] += watermarkedPixels[i] & secretBit;
    }
} else {
    if ((watermarkedPixels[i] >> 1) % 2 == 0) {
        watermarkedPixels[i] |= secretBit << 1;
    } else {
        watermarkedPixels[i] = (((watermarkedPixels[i] >> 1) &
(secretBit)) << 1) + 1;
    }
}
}

```

Діставання коду з зображення

```

private String getSecret(Bitmap bitmap, String key) {
    int width = bitmap.getWidth();
    int height = bitmap.getHeight();
    int[] buffer = new int[width * height];
    bitmap.getPixels(buffer, 0, width, 0, 0, width, height);

    String keyBinary = stringToBinary(key);
    StringBuilder resultBinary = new StringBuilder();

    for (int i = 0; i < keyBinary.length(); i++) {
        int secretBit = keyBinary.charAt(i) == '0' ? buffer[i] & 1 : ((buffer[i]
& 0b10) >> 1);
        resultBinary.append(secretBit == 1 ? "1" : "0");
    }

    return binaryStringToNormal(resultBinary.toString());
}

```

Наразі діставання секретного коду не працює через незрозумілі технічні проблеми, але записування відбувається справно.

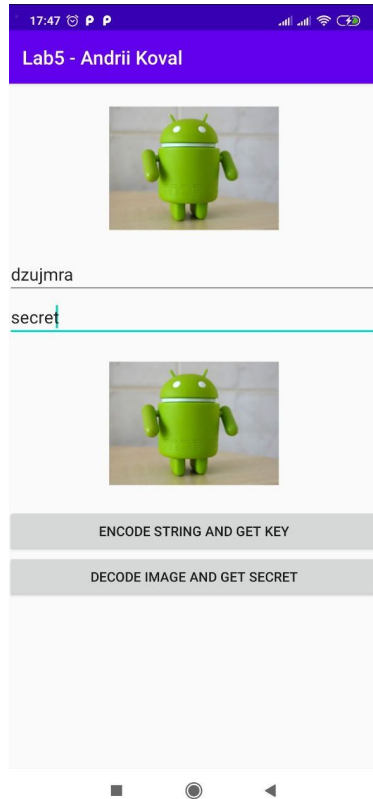


Рис 7, Зовнішній вигляд завдання 6

Висновки

Я ознайомився із основними принципами обробки графічних даних з використанням основних методів і алгоритмів цифрової обробки зображень. Ці знання неодмінно допоможуть у майбутньому не тільки в Android розробці, а й у будь якій іншій галузі, оскільки тема цифрової обробки зображень є платформонезалежною.