КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
ФАКУЛЬТЕТ КОМП’ЮТЕРНИЙ НАУК ТА КІБЕРНЕТИКИ

Звіт
до лабораторної роботи
"Визначення (знаходження) облич людей на сцені."
з курсу
“Розпізнавання образів та машинне навчання”

Виконав студент групи ТК-4
Вернигор Віталій Вадимович

Київ
2021

Постановка задачі

Лабораторний практикум передбачає створення системи для знаходження на картинці облич людей.

Реалізація

1. Встановлення.

Виконання лабораторної було почато з встановлення бібліотеки OpenCV Java. Для цього було застосовано Apache Maven - фреймворк для автоматизації збирання проектів на основі опису їх структури в файлах на мові POM, що є підмножиною XML. На скриншоті нижче наведено які саме залежності використовувалися:
pom.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>ROAS</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>org.openpnp</groupId>
            <artifactId>opencv</artifactId>
            <version>3.4.2-0</version>
        </dependency>
    </dependencies>

    <properties>
        <maven.compiler.source>15</maven.compiler.source>
        <maven.compiler.target>15</maven.compiler.target>
    </properties>

</project>
```

Це дозволило полегшити запуск мого проекту на інших комп’ютерах.

2. Застосування бібліотеки OpenCV.

Далі в основному файлі проекту “Main.java” я ініціалізував бібліотеку OpenCV в методі “main”:

```java
public class Main {
    public static void main(String[] args) {

        OpenCV.loadShared();
        System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
```

Також я імпортував такі пакети:

```java
import nu.pattern.OpenCV;
import org.opencv.core.*;
import org.opencv.imgcodecs.Imgcodecs;
import org.opencv.imgproc.Imgproc;
import org.opencv.objdetect.CascadeClassifier;
import org.opencv.objdetect.Objdetect;
```

3. Завантаження та збереження зображень.

Далі я додав методи завантаження та збереження зображень:

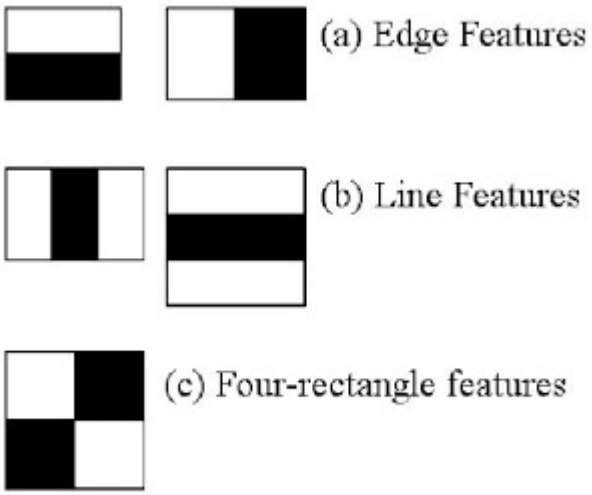
```java
    public static Mat loadImage(String imagePath) {
        Imgcodecs imageCodecs = new Imgcodecs();
        return imageCodecs.imread(imagePath);
    }

    public static void saveImage(Mat imageMatrix, String targetPath) {
        Imgcodecs imgcodecs = new Imgcodecs();
        imgcodecs.imwrite(targetPath, imageMatrix);
    }
```

Метод “loadImage” зчитує зображення з диску за переданою в нього адресою та повертає у місце виклику об’єкт типу Mat, який є матричним поданням зображення. Метод “saveImage” записує матричне подання обробленого файлу зображення на диск.

4. Haar Cascade Classifier.

Далі мною було підключено каскадний класифікатор Haar.
Класифікатор - це програма, яка прагне розмістити нове спостереження в групі, яка залежить від минулого досвіду. Каскадні класифікатори прагнуть зробити це за допомогою об'єднання кількох класифікаторів. Кожен наступний класифікатор використовує вихідні дані попереднього як додаткову інформацію, значно покращуючи класифікацію.
Розпізнавання обличчя в OpenCV здійснюється за допомогою каскадних класифікаторів на основі Haar.

Haar-функції - це фільтри, які використовуються для виявлення країв та ліній на зображенні. Фільтри розглядаються як квадрати з чорно-білими кольорами:



Ці фільтри застосовуються кілька разів до зображення, піксель за пікселем, і результат збирається як одне значення. Це значення - різниця між сумою пікселів під чорним квадратом та сумою пікселів під білим квадратом.

Як правило, каскадний класифікатор повинен бути попередньо навчений, щоб мати можливість виявляти що-небудь взагалі.

5. Знаходження обличчя.

Оскільки навчальний процес може бути тривалим і вимагатиме великого набору даних, я використав одну з попередньо навчених моделей, запропонованих OpenCV.
Для зручності цей файл XML було розміщено у папці ресурсів. Путь із коріння проекту виглядає так: “src/main/resources/haarcascade_frontalface_alt.xml”.

Ми спробуємо виявити обличчя, окресливши його червоним прямокутником.

Для початку потрібно завантажити зображення у форматі Mat з нашого вихідного шляху:

```
18          Mat loadedImage = loadImage(sourceImagePath);
```

Потім я оголосив об’єкт MatOfRect для зберігання знайдених облич:

```
20          MatOfRect facesDetected = new MatOfRect();
```

Далі потрібно було ініціалізувати CascadeClassifier для розпізнавання:

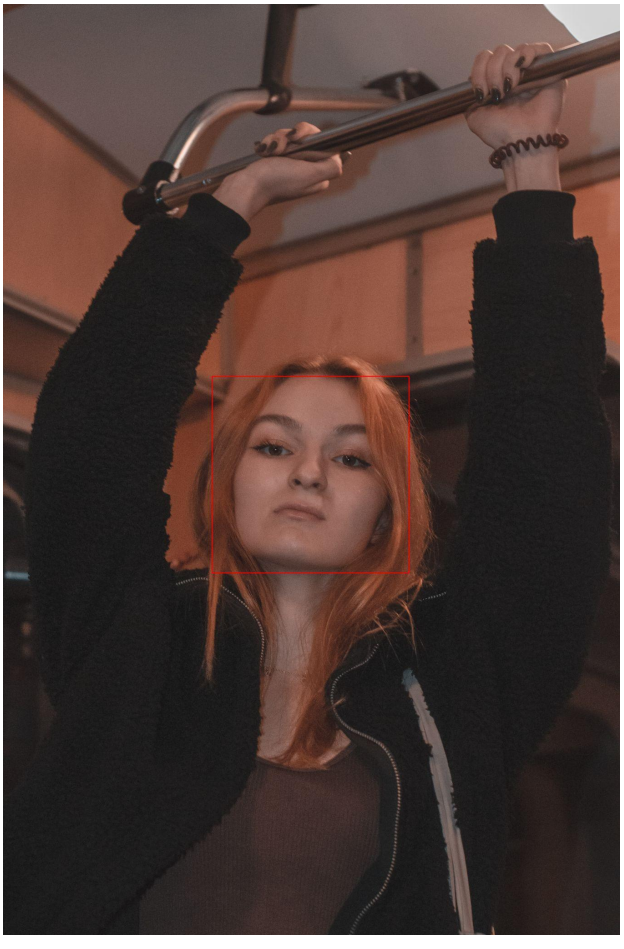
```
CascadeClassifier cascadeClassifier = new CascadeClassifier();
int minFaceSize = Math.round(loadedImage.rows() * 0.1f);
cascadeClassifier.load( filename: "src/main/resources/haarcascade_frontalface_alt.xml");
cascadeClassifier.detectMultiScale(loadedImage,
        facesDetected,
        scaleFactor: 1.1,
        minNeighbors: 3,
        Objdetect.CASCADE_SCALE_IMAGE,
        new Size(minFaceSize, minFaceSize),
        new Size()
);
```

Вище параметр 1.1 позначає масштабний коефіцієнт, який ми хочемо використовувати, вказуючи, наскільки зменшений розмір зображення при кожному масштабі зображення. Наступним параметром, 3, є minNeighbors. Це кількість сусідів, яку повинен мати прямокутник-кандидат, щоб зберегти його.

Нарешті, ми запустимо цикл та збережемо результат:

```
Rect[] facesArray = facesDetected.toArray();
for(Rect face : facesArray) {
    Imgproc.rectangle(loadedImage, face.tl(), face.br(), new Scalar(0, 0, 255),  thickness: 3);
}
saveImage(loadedImage, targetImagePath);
```

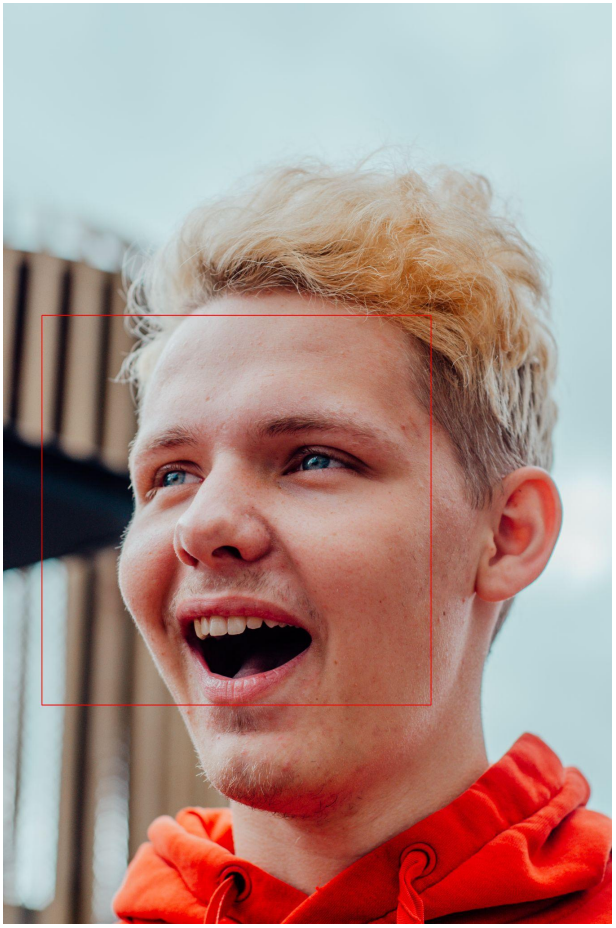
Зараз ввевши вихідне зображення, ми повинні отримати вихідне зображення з усіма гранями, позначеними червоним прямокутником.
Нижче наведено декілька прикладів вихідних зображень.
Автором кожного з ісходних зображень є я, Вернигор Віталій. При Використанні цих зображень нічиїх прав порушено не було.



Приклади вихідних зображень

Перше зображення.

Найпростіший приклад для розпізнавання обличчя.
На зображенні людина лицем до камери (Анфас), обличчя рівномірно освітлене

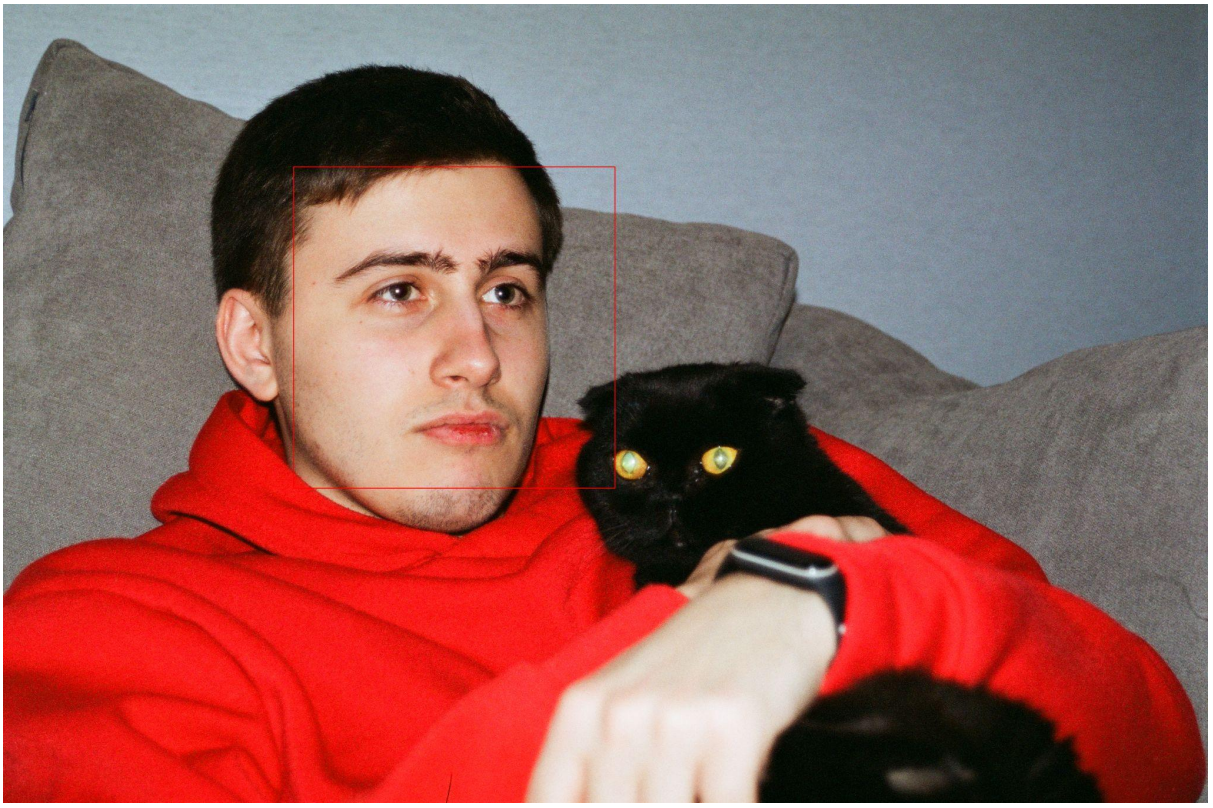


Друге зображення.

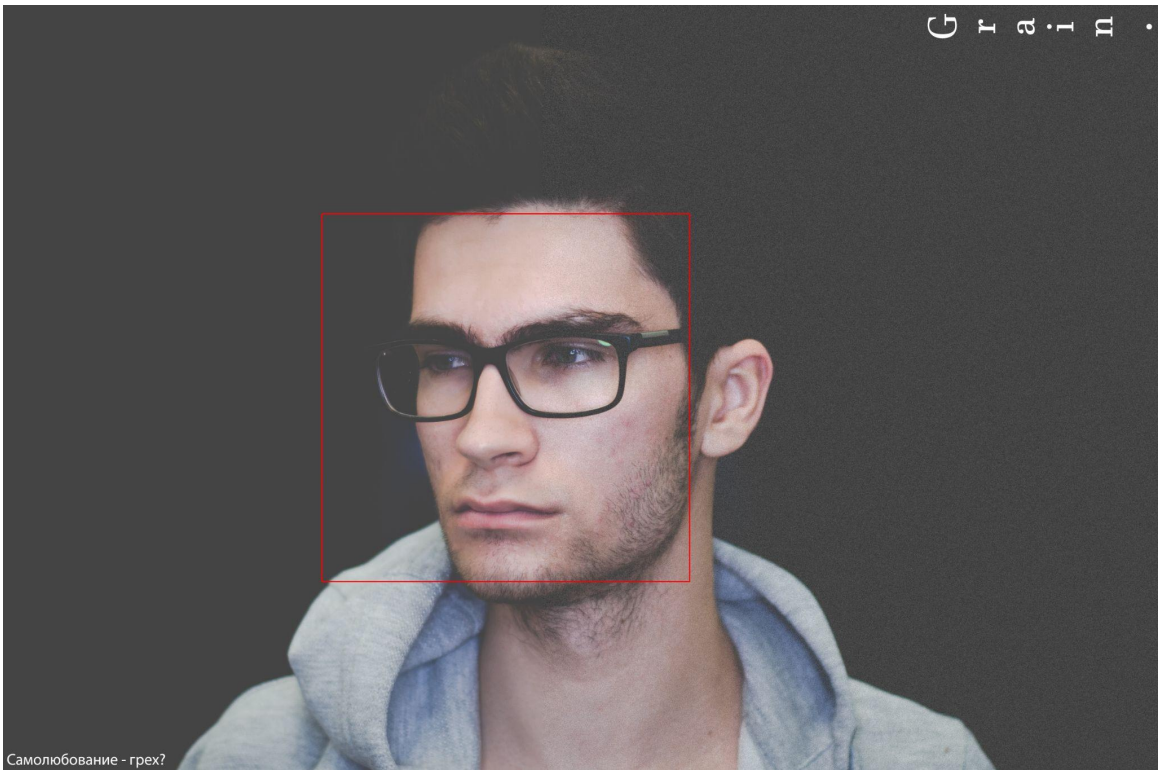
Це зображення складніше за попереднє, бо людина звернена до камери під кутом приблизно 45 градусів. Але-ж, як бачимо, алгоритм впорався.

Третє зображення.

Було спробувано заплутати алгоритм додавши морду кішки, але алгоритм впорався



Четверте зображення.



Тут я спробував ускладнити життя алгоритму, додавши шуму на частину зображення, та обравши своє фото в окулярах, але ж і з цим алгоритм впорався.

П’яте зображення.

Тут я перевірив чи коректно працює алгоритм, якщо на сцені кілька облич одночасно.

Шосте зображення

Найскладніша, на мою думку, композиція для алгоритму, бо частини обличчя знаходяться у тіні. Але ж і на цьому зображенні алгоритм у змозі знайти обличчя.

Сьоме зображення. Невдале.

Як виявилось запропонований OpenCV каскадний класіфікатор не вміє розпізнавати обличчя в захисній масці.

Висновок

Мною було розроблено систему пошуку обличчя на сцені. Для цього було використано бібліотеку алгоритмів комп'ютерного зору, обробки зображень та чисельних алгоритмів загального призначення з відкритим кодом OpenCV, мову програмування Java, фреймворк для автоматизації збирання проектів Apache Maven.

Із плюсів використання бібліотеки OpenCV зазначено: проста та швидка установка, добре реалізована інтеграція з мовами програмування, швидкість роботи.

Із мінусів можна визначити тільки невеликий об’єм вже навченних класифікаторів. (В реаліях епідемії потрібно навчати свої системи розпізнавати людей в масках.)

Література:

1. G. Bradski, A. Kaehler Learning OpenCV: Computer Vision with the OpenCV Library.– O'Reilly Media, Inc., 2008.– 580 p.
2. Кэлер А., Брэдски Г. Изучаем OpenCV. — М.: ДМК-Пресс, 2017. — 826 с.
3. Буэно, Суарес, Эспиноса. Обработка изображений с помощью OpenCV = Learning Image Processing with OpenCV. — М.: ДМК-Пресс, 2016. — 210 с.
4. Прохоренок Н. А. OpenCV и Java. Обработка изображений и компьютерное зрение. — СПб.: БХВ-Петербург, 2018. — 320 с.: ил.

Електронні ресурси:

http://opencv.willowgarage.com/documentation/cpp/index.html
https://opencv.org/
https://docs.opencv.org/
http://robocraft.ru/tag/OpenCV/

https://github.com/Taska23/ROAS
https://t.me/Taska2399