Guilherme Parpineli

BackEnd Developer

Reconhecimento facial com java

Arraste para mais

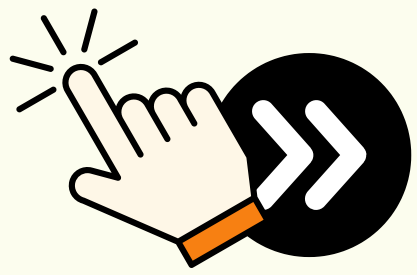
Guilherme Parpineli

BackEnd Developer

OpenCV

OpenCV é a maior biblioteca para visão computacional do mundo com mais de 2500 algoritmos disponíveis e ainda open source. Vamos fazer uma implementação básica com Java 25 e Spring Boot e explicar os conceitos básicos.

Arraste para mais



Guilherme Parpineli BackEnd Developer

Setup inicial

Para iniciar o projeto, você irá precisar ter JDK 25 uma boa IDE e uma CPU decente. Lembrando que funciona em JDKs mais antigas até mesmo no Java 8.

Project

- ○ Gradle - Groovy
- ● Gradle - Kotlin
- ○ Maven

Language

- ● Java
- ○ Kotlin
- ○ Groovy

Spring Boot

- ○ 4.0.2 (SNAPSHOT)
- ● 4.0.1
- ○ 3.5.10 (SNAPSHOT)
- ○ 3.5.9

Project Metadata

Group: com.example
Artifact: opencv-demo
Name: opencv-demo
Description: Demo project for Spring Boot
Package name: com.example.opencv-demo
Packaging: ● Jar ○ War
Configuration: ○ Properties ● YAML
Java: ● 25 ○ 21 ○ 17

Dependencies ADD DEPENDENCIES... ⌘ + B

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Arraste para mais

Guilherme Parpineli BackEnd Developer

Dependências

Para simplificar o desenvolvimento podemos adicionar a dependência no gradle abaixo:

implementation("org.bytedeco:opencv-platform:4.11.0-1.5.12")

Esta dependência é essencial porque o OpenCV é desenvolvido em C++ e requer um wrapper para ser utilizado em Java. O diferencial desta biblioteca em relação à implementação oficial do OpenCV é a sua portabilidade: ela dispensa a instalação de binários no sistema operacional e utiliza o JavaCPP para otimizar o gerenciamento de memória.

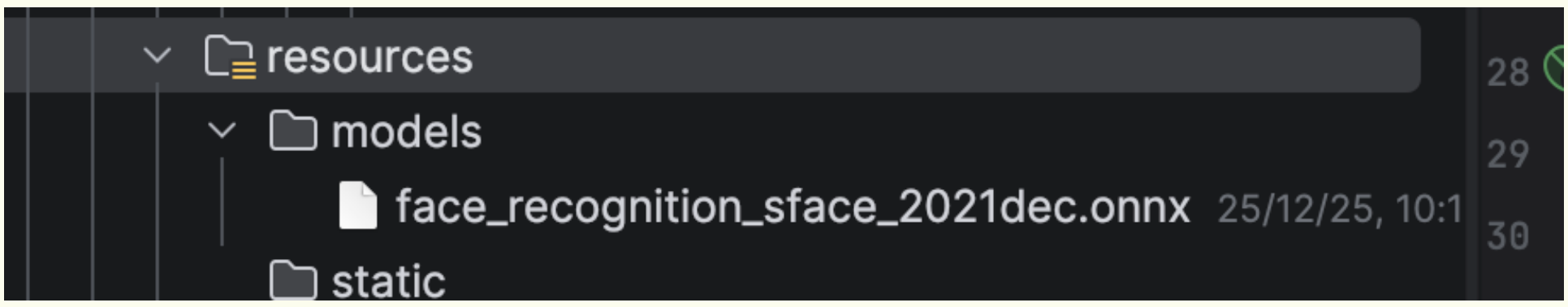
Irei explicar melhor sobre JNI, JNA, JavaCPP em outro post

Arraste para mais

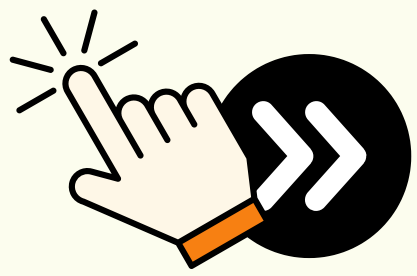
Guilherme Parpineli BackEnd Developer

Configurando modelo

Há diversos modelos disponíveis, neste caso iremos utilizar um modelo Deep Learning chamado SFace. Para isso baixe o modelo e coloque na resources de seu projeto(link na descrição)



Arraste para mais



Guilherme Parpineli BackEnd Developer

Service

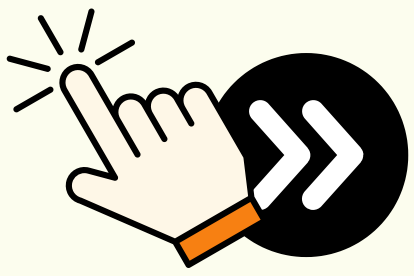
```
@Service  3 usages
public class RecognitionService {

    String sFaceModel = extractModelToTempFile();
```

Neste exemplo eu extrai para um arquivo temp porém o mais adequado é possuir o arquivo de modelo no servidor e aqui receber o path do modelo para evitar toda vez fazer o msm processo.

```
static {
    Loader.load(opencv_java.class);
}
```

Carregamos a biblioteca nativa do OpenCV disposta na bytedeco.

Arraste para mais



Guilherme Parpineli BackEnd Developer

```
private final FaceRecognizerSF recognizer = FaceRecognizerSF.create(  3 usa
        sFaceModel,  config: "", Dnn.DNN_BACKEND_OPENCV, Dnn.DNN_TARGET_CPU
);
```

Nesta etapa, instanciei a classe principal FaceRecognizerSF, pertencente à biblioteca OpenCV, para realizar o carregamento do modelo. O primeiro parâmetro define o caminho (path) do arquivo; o segundo refere-se às configurações que, por se tratar de um modelo moderno, já estão integradas ao próprio arquivo. O terceiro parâmetro especifica o motor de execução (backend), onde optei pelo nativo do OpenCV, embora haja suporte para OpenVINO e CUDA. Por fim, o quarto parâmetro define o hardware de processamento, podendo ser CPU ou GPU (via CUDA).

Arraste para mais

Guilherme Parpineli BackEnd Developer

```java
public Mat extractFeatures(Mat origImage, Mat faceImage) {  2 us
    Mat targetAligned = new Mat();
    recognizer.alignCrop(origImage, faceImage, targetAligned);
    Mat targetFeatures = new Mat();
    recognizer.feature(targetAligned, targetFeatures);
    return targetFeatures.clone();
}
```

Este método é responsável por extrair o vetor de características (feature vector) da imagem. Inicialmente, aplicamos o alignCrop para normalizar a face; isso é crucial, pois variações de inclinação, distância ou posicionamento dificultam a comparação biométrica. Após o alinhamento, o método feature gera um embedding: uma matriz (geralmente de 1×128 ou 1×512 elementos) que serve como uma representação numérica abstrata e única da identidade da pessoa.

Arraste para mais

Guilherme Parpineli BackEnd Developer

```java
public MatchResult matchFeatures(Mat targetFeatures, Mat queryFeatures) {  1 usage
    int distanceType = 0;
    double THRESHOLD_COSINE = 0.36;
    double score = recognizer.match(targetFeatures, queryFeatures, distanceType);
    boolean isMatch;
    isMatch = score ⩾ THRESHOLD_COSINE;
    return new MatchResult(score, isMatch);
}
```

Este método executa a etapa final de reconhecimento facial. O parâmetro distanceType define a métrica matemática utilizada para comparar os vetores de características (embeddings). No modelo SFace do OpenCV, o valor 0 especifica a Similaridade de Cosseno (FR_COSINE). Esta métrica calcula o cosseno do ângulo entre os vetores, sendo particularmente eficaz em mitigar variações de iluminação. O THRESHOLD_COSINE define o limiar de decisão: se o resultado da similaridade for igual ou superior a este valor, a correspondência é validada como positiva.

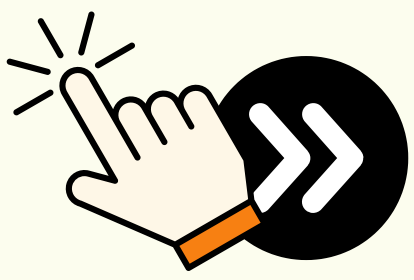
Arraste para mais

Guilherme Parpineli BackEnd Developer

```java
@PostMapping(consumes = MediaType.MULTIPART_FORM_DATA_VALUE)  Guilherme Parpineli
public ResponseEntity<MatchResult> compareImages(
        @RequestParam("targetImage") MultipartFile targetImage,
        @RequestParam("queryImage") MultipartFile queryImage
) {
    Attempts image matching; returns error on failure
    try {
        Mat target = loadFromBytesInMemory(targetImage.getBytes(), targetImage.getOriginalFilename());
        Mat query = loadFromBytesInMemory(queryImage.getBytes(), queryImage.getOriginalFilename());
        Mat queryFeatures = service.extractFeatures(query, query);
        Mat targetFeatures = service.extractFeatures(target, target);
        return ResponseEntity.ok(service.matchFeatures(targetFeatures, queryFeatures));
    } catch (IOException e) {
        return ResponseEntity.badRequest().body(new MatchResult( score: -1,  isMatch: false));
    }
}
```

Para finalizar, apresento a Controller. A arquitetura foi simplificada para este exemplo, expondo um único endpoint de processamento. Nele, recebemos os arquivos via MultipartFile e realizamos a conversão para o objeto Mat (matriz nativa do OpenCV), permitindo que a imagem entre no pipeline de reconhecimento. Por ser um processo de integração mais comum, o código completo de conversão e as rotas estão detalhados no repositório do GitHub.

Arraste para mais

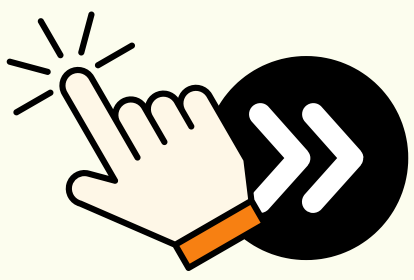


Guilherme Parpineli BackEnd Developer

Este fluxo demonstra como a Visão Computacional moderna evoluiu: transformamos pixels em geometria com e geometria em álgebra linear. O reconhecimento facial, no fim das contas, é uma grande comparação matemática de ângulos em um espaço de alta dimensão.

Link do repositório de modelo na descrição.

Arraste para mais



Guilherme Parpineli

BackEnd Developer

Se você achou isso útil, por favor reaja e compartilhe com sua rede

@guiparpineli