Machine Learning - Desenvolvendo uma API de Classificacao de Imagens em Java

Este artigo apresenta o desenvolvimento de uma API para classificação de imagens utilizando **Spring Boot** e **Deep Java Library (DJL)**, organizado em uma estrutura **MVC (Model-View-Controller)**.

A abordagem fornece uma arquitetura performática e escalável para aplicações que necessitam de alta capacidade de resposta.

Usaremos o DJL, uma biblioteca de Machine Learning em Java, utilizando o modelo **ResNet50** para classificação de imagens.

Este projeto é ideal para profissionais que buscam uma integração de Machine Learning em suas soluções Java, com uma implementação flexível e de alta performance.

Configurações Iniciais

- **Versão do Java**: Java 11 ou superior (Java 17 é recomendada para otimização e compatibilidade com bibliotecas recentes).
- **Bibliotecas utilizadas**: DJL para modelagem e Spring Boot para a API.

Dependências

Adicione as dependências no arquivo pom.xml para configurar o Spring Boot e a DJL:

```
<dependencies>
    <!-- Spring Boot Web para API REST -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <!-- Dependências DJL para suporte a MXNet -->
    <dependency>
        <groupId>ai.djl</groupId>
        <artifactId>api</artifactId>
        <version>0.20.0</version>
    </dependency>
    <dependency>
        <groupId>ai.djl.mxnet</groupId>
        <artifactId>mxnet-engine</artifactId>
        <version>0.20.0</version>
    </dependency>
```

```
    <dependency>
        <groupId>ai.djl.mxnet</groupId>
        <artifactId>mxnet-model-zoo</artifactId>
        <version>0.20.0</version>
    </dependency>
</dependencies>
```

Estrutura da Aplicação

A aplicação está estruturada conforme a arquitetura MVC:

1. **Modelo**: Define a estrutura da resposta da API.
2. **Serviço**: Contém a lógica de carregamento e execução do modelo de Machine Learning.
3. **Controlador**: Gerencia o endpoint da API e as requisições HTTP.
4. **Configuração (Opcional)**: Carrega e gerencia o modelo para otimizar recursos.

1. Modelo Prediction

A classe Prediction representa a estrutura da resposta. Cada instância desta classe inclui a **classe da predição** e a **probabilidade** associada.

```
package com.example.mlapi.model;

/**
 * Modelo de dados para representar uma predição de classificação de imagem.
 */
public class Prediction {
    private final String className;  // Nome da classe identificada (e.g., "Cachorro")
    private final double probability; // Probabilidade de pertencer a essa classe

    // Construtor para inicializar atributos
    public Prediction(String className, double probability) {
        this.className = className;
        this.probability = probability;
    }

    // Getters para acesso aos atributos
    public String getClassName() {
        return className;
    }

    public double getProbability() {
        return probability;
    }
}
```

2. Serviço ImageClassificationService

A classe ImageClassificationService realiza a lógica de carregamento do modelo e classificação das imagens. É projetada para ser altamente performática e reutilizar o modelo carregado.

```
package com.example.mlapi.service;
```

```
import ai.djl.Model;
import ai.djl.ModelException;
import ai.djl.modality.Classifications;
import ai.djl.modality.cv.Image;
import ai.djl.modality.cv.ImageFactory;
import ai.djl.translate.TranslateException;
import com.example.mlapi.model.Prediction;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;
import java.util.List;
import java.util.stream.Collectors;

/**
 * Serviço que realiza a classificação de imagens com DJL.
 */
@Service
public class ImageClassificationService {

    private final Model model;

    /**
     * Construtor que injeta o modelo de machine learning para reutilização.
     */
    @Autowired
    public ImageClassificationService(Model model) {
        this.model = model;
    }

    /**
     * Classifica a imagem enviada e retorna uma lista de predições.
     *
     * @param file Imagem enviada via upload
     * @return Lista de predições com a classe e probabilidade
     */
    public List<Prediction> classifyImage(MultipartFile file) throws IOException, ModelException,
TranslateException {
        // Processa a imagem no formato DJL para compatibilidade com o modelo
        Image image = ImageFactory.getInstance().fromInputStream(file.getInputStream());

        // Realiza a predição de maneira performática
        Classifications classifications = model.predict(image);

        // Converte os resultados para uma lista de objetos Prediction
        return classifications.items().parallelStream()
                .map(item -> new Prediction(item.getClassName(), item.getProbability()))
                .collect(Collectors.toList());
    }
}
```

3. Controlador ImageClassificationController

O controlador expõe o **endpoint REST** /api/classify, onde os clientes podem enviar uma imagem para classificação.

```
package com.example.mlapi.controller;

import com.example.mlapi.model.Prediction;
import com.example.mlapi.service.ImageClassificationService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;

import java.util.List;

/**
 * Controlador REST para expor o endpoint de classificação de imagens.
 */
@RestController
@RequestMapping("/api/classify")
public class ImageClassificationController {

    private final ImageClassificationService classificationService;

    @Autowired
    public ImageClassificationController(ImageClassificationService classificationService) {
        this.classificationService = classificationService;
    }

    /**
     * Endpoint que recebe uma imagem e retorna as predições de classificação.
     *
     * @param file Imagem enviada como multipart/form-data
     * @return Lista de predições com as classes e probabilidades
     */
    @PostMapping
    public ResponseEntity<List<Prediction>> classifyImage(@RequestParam("image")
MultipartFile file) {
        try {
            List<Prediction> predictions = classificationService.classifyImage(file);
            return ResponseEntity.ok(predictions);
        } catch (Exception e) {
            // Log e retorno de erro
            e.printStackTrace();
            return ResponseEntity.status(500).body(null);
        }
    }
}
```

4. Configuração ModelConfig (Opcional)

A configuração do modelo DJL usa a classe ModelConfig, permitindo que o modelo seja carregado uma única vez.

```
package com.example.mlapi.config;

import ai.djl.Model;
import ai.djl.ModelException;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
 * Configuração para carregar o modelo DJL apenas uma vez.
 */
@Configuration
public class ModelConfig {

    @Bean
    public Model model() throws ModelException {
        return Model.newInstance("resnet50");
    }
}
```

Aplicações Potenciais da Machine Learning API

Esta API de classificação de imagens pode ser usada em várias áreas:

1. **Segurança e Vigilância**: Identificação em tempo real de objetos ou pessoas.
2. **E-commerce**: Categorização automática de produtos a partir de imagens.
3. **Assistência Médica**: Diagnóstico de condições médicas em imagens.
4. **Agronegócio**: Análise de espécies vegetais e condições do solo.
5. **Social Media**: Moderação de conteúdo e categorização em redes sociais.

Exemplo de Requisição HTTP e Resposta JSON

Para enviar uma imagem e receber a classificação, faça uma requisição **POST** com o seguinte formato:

```
curl -X POST http://localhost:8080/api/classify \
  -F image=@/caminho/para/sua/imagem.jpg
```

Exemplo de Resposta JSON


```
[
  {
    "className": "Cachorro",
    "probability": 0.85
  },
  {
    "className": "Gato",
    "probability": 0.1
  },
  {
    "className": "Pássaro",
    "probability": 0.05
  }
]
```


Conclusão

Usar uma estrutura de **API performática em arquitetura MVC** com Spring Boot e DJL permite uma organização eficiente do código e alta escalabilidade.

Esse projeto é altamente modular e adaptável para várias aplicações reais, sendo ideal para integrar Machine Learning em Java de forma prática e performática.

EducaCiência FastCode para comunidade