Machine Learning - Classificacao de Imagens e treinamento

Este guia aborda a criação de uma API para classificação de imagens usando Machine Learning, Java, e Spring Boot, com uma arquitetura robusta MVC (Model-View-Controller).

Inclui o detalhamento de como configurar e treinar o modelo, com exemplos de endpoints e responses da API, além de aplicações práticas para Machine Learning.

Estrutura Completa do Artigo

1. Configuração do Ambiente e Dependências
2. Estrutura da Arquitetura MVC
3. Implementação Completa do Modelo de Machine Learning
4. Treinamento do Modelo com DJL
5. Endpoints da API e Exemplo de Response JSON
6. Exemplos de Aplicação do Modelo de Machine Learning
7. Conclusão

Configuração do Ambiente e Dependências

Para este projeto, usaremos Java (versões 8, 11 ou 17), Spring Boot e a DJL (Deep Java Library) para machine learning. O pom.xml incluirá todas as dependências necessárias para a biblioteca DJL e Spring Boot.

Dependências no pom.xml

```
<dependencies>
    <!-- Dependência do Spring Boot -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
        <version>2.6.3</version>
    </dependency>
    <!-- Dependências para DJL -->
    <dependency>
        <groupId>ai.djl</groupId>
        <artifactId>api</artifactId>
        <version>0.20.0</version>
    </dependency>
```

```
    <dependency>
        <groupId>ai.djl.training</groupId>
        <artifactId>training</artifactId>
        <version>0.20.0</version>
    </dependency>
    <dependency>
        <groupId>ai.djl.mxnet</groupId>
        <artifactId>mxnet-engine</artifactId>
        <version>0.20.0</version>
    </dependency>
</dependencies>
```

Estrutura da Arquitetura MVC

Dividiremos a aplicação em três camadas: **Controller** (Controlador), **Service** (Serviço) e **Model** (Modelo).

Controlador (ImageClassificationController)

Esta classe é responsável por receber e processar as requisições HTTP.

```
import org.springframework.web.bind.annotation.*;
import org.springframework.http.ResponseEntity;

@RestController
@RequestMapping("/api/classify")
public class ImageClassificationController {

    private final ImageClassificationService classificationService;

    // Injeção de dependência do serviço
    public ImageClassificationController(ImageClassificationService classificationService) {
        this.classificationService = classificationService;
    }

    // Endpoint POST para classificar a imagem
    @PostMapping("/predict")
    public ResponseEntity<PredictionResponse> predictImage(@RequestBody ImageRequest
imageRequest) {
        // Chama o serviço para classificar a imagem e gera a resposta com o resultado da
predição
        String prediction = classificationService.classifyImage(imageRequest);
        return ResponseEntity.ok(new PredictionResponse(prediction));
    }
}
```

Serviço (ImageClassificationService)

Essa camada implementa a lógica de negócios, conectando-se ao modelo de Machine Learning para realizar a classificação.

```
import ai.djl.Model;
import ai.djl.inference.Predictor;
```

```
import ai.djl.translate.TranslateException;
import org.springframework.stereotype.Service;

@Service
public class ImageClassificationService {

    private final Model model;

    // Construtor para carregar o modelo de Machine Learning
    public ImageClassificationService() throws IOException {
        model = Model.newInstance("image-classification");
        model.load(Paths.get("model"));
    }

    // Método para classificar a imagem e retornar o resultado
    public String classifyImage(ImageRequest imageRequest) {
        try (Predictor<Image, Classifications> predictor = model.newPredictor()) {
            return predictor.predict(imageRequest.getImage()).toString();
        } catch (TranslateException e) {
            throw new RuntimeException("Erro ao classificar a imagem", e);
        }
    }
}
```

Modelo (ImageRequest e PredictionResponse)

Essas classes representam a estrutura de dados de entrada e saída para a API.

```
// Classe para receber a imagem como entrada da API
public class ImageRequest {
    private Image image;

    // Getters e Setters
}

// Classe para encapsular a resposta da predição
public class PredictionResponse {
    private String prediction;

    public PredictionResponse(String prediction) {
        this.prediction = prediction;
    }

    // Getter para obter a predição
    public String getPrediction() {
        return prediction;
    }
}
```

Implementação Completa do Modelo de Machine Learning

Usamos uma rede **ResNet50** para a classificação.

Esse modelo é pré-treinado e pode ser carregado diretamente para produção, garantindo alta precisão e performance.

Treinamento do Modelo com DJL

Preparação do Dataset

O dataset deve ser estruturado com subpastas para cada classe, por exemplo:

```
dataset/
├── cachorros/
│   ├── dog1.jpg
│   ├── dog2.jpg
└── gatos/
    ├── cat1.jpg
    ├── cat2.jpg
```

Código para Treinamento

```
import ai.djl.Model;
import ai.djl.basicdataset.cv.classification.ImageFolder;
import ai.djl.modality.cv.transform.Resize;
import ai.djl.modality.cv.transform.ToTensor;
import ai.djl.training.Trainer;
import ai.djl.training.dataset.Batch;
import ai.djl.training.listener.LoggingTrainingListener;
import ai.djl.training.loss.Loss;

import java.nio.file.Paths;

public class ImageClassificationTraining {

    public static void main(String[] args) throws Exception {
        ImageFolder dataset = ImageFolder.builder()
                .setRepositoryPath(Paths.get("dataset"))
                .optPipeline(new Resize(224), new ToTensor())
                .setSampling(32, true)
                .build();

        dataset.prepare(new ProgressBar());

        try (Model model = Model.newInstance("image-classification")) {
            model.setBlock(ResNet50.builder().build());

            Trainer trainer = model.newTrainer(new
DefaultTrainingConfig(Loss.softmaxCrossEntropyLoss())
                    .addEvaluator(new Accuracy())
```

```
                .addTrainingListeners(new LoggingTrainingListener()));

            // Loop de treinamento por 10 épocas
            for (int epoch = 0; epoch < 10; epoch++) {
                for (Batch batch : trainer.iterateDataset(dataset)) {
                    EasyTrain.trainBatch(trainer, batch);
                    trainer.step();
                    batch.close();
                }
                trainer.endEpoch();
            }

            model.save(Paths.get("model"), "image-classification");
        }
    }
}
```

Exemplo de Request

```
POST /api/classify/predict
{
  "image": "base64stringdaimagem"
}
```

Exemplo de Response

```
{
  "prediction": "cachorro",
  "confidence": 0.95
}
```

Exemplos de Aplicação do Modelo de Machine Learning

- **Classificação de Documentos**: Aplicável a documentos digitalizados para organizar automaticamente arquivos em categorias.
- **Monitoramento de Fauna**: Classificação de imagens em sistemas de câmeras em áreas florestais para monitorar espécies de animais.
- **Identificação de Espécies de Plantas e Animais**: Para aplicativos educacionais e científicos de reconhecimento de espécies.
- **Análise de Segurança**: Identificação de imagens de segurança em câmeras para categorização e análises posteriores.

Este artigo explora a implementação de uma API de Machine Learning em Java com Spring Boot e DJL, usando uma arquitetura MVC. A abordagem com código altamente performático permite que a aplicação seja facilmente adaptada a diferentes cenários de classificação de imagens, oferecendo flexibilidade e escalabilidade.

EducaCiência FastCode para comunidade