Python vers Java

TensorFlow ou PyTorch

Objectif :

L'objectif de ces heures est l'explication de comment exporter un réseau de neurones entraîné en Python avec Tensorflow ou Pytorch en Java pour pouvoir l'utiliser dans votre projet de bureau d'études de vision par ordinateur. Vous avez plusieurs choix d'exportation HDF5, ONNX, … ou aussi l'utilisation de TensorFlow Java API.

1. Exporter en format HDF5 (pour les modèles TensorFlow/Keras)

Si vous avez utilisé **TensorFlow** ou **Keras**, vous pouvez enregistrer le modèle sous forme de fichier .h5 (format HDF5), et ensuite l'utiliser avec Java.

Étapes :

1) **Enregistrer le modèle en Python** :

```
model.save("model.h5") # format HDF5
# ou
model.save("saved_model") # format SavedModel
```

2) **Charger le modèle dans DeepLearning4J**

Ajouter les dépendances Maven :

```
<dependency>
   <groupId>org.deeplearning4j</groupId>
   <artifactId>deeplearning4j-core</artifactId>
   <version>1.0.0-M2.1</version>
</dependency>

<dependency>
   <groupId>org.nd4j</groupId>
   <artifactId>nd4j-native-platform</artifactId>
   <version>1.0.0-M2.1</version>
</dependency>
```

3) **Charger et utiliser ce modèle en Java** :

Utiliser une bibliothèque Java comme **DeepLearning4J** (DL4J) qui prend en charge l'importation de modèles Keras et TensorFlow.

Exemple avec DL4J :

```
import org.deeplearning4j.nn.graph.ComputationGraph;
import org.deeplearning4j.util.ModelSerializer;

public class LoadModel {
    public static void main(String[] args) throws Exception {
        ComputationGraph model = ModelSerializer.restoreComputationGraph("model.h5");
        // Utilisez le modèle pour effectuer des prédictions
    }
}
```

Autre exemple avec DL4J :

```
import org.deeplearning4j.nn.modelimport.keras.KerasModelImport;
import org.deeplearning4j.nn.graph.ComputationGraph;

public class LoadModel {

    public static void main(String[] args) throws Exception {

        String modelPath = "model.h5";

        ComputationGraph model = KerasModelImport
                .importKerasModelAndWeights(modelPath);

        System.out.println("Modèle chargé !");
    }
}
```

Exemple avec DL4J d'un cas de modèle TensorFlow avec « saved_model » :

```
import org.deeplearning4j.nn.graph.ComputationGraph;
import org.deeplearning4j.nn.modelimport.tensorflow.TFGraphMapper;

ComputationGraph model = TFGraphMapper.importGraph(new File("saved_model"));
```

4) **Faire une prédiction**

```
import org.nd4j.linalg.factory.Nd4j;
import org.nd4j.linalg.api.ndarray.INDArray;

INDArray input = Nd4j.create(new float[]{...}, new int[]{1, inputSize});
```

```
INDArray output = model.outputSingle(input);

System.out.println(output);
```

Attention :

L'import TensorFlow natif est parfois limité selon :

- Les opérations utilisées
- Les couches personnalisées
- Les fonctions Lambda
- Les custom layers

Limitations importantes :

DL4J ne supporte pas :

- Les couches custom TensorFlow
- Certaines opérations récentes TF 2.x
- Les modèles très récents type Transformers avancés

2. Exporter en utilisant ONNX (Open Neural Network Exchange) (Architecture recommandée)

ONNX est un format de fichier ouvert qui permet de transférer des modèles entre différents frameworks de machine learning. Il est pris en charge par de nombreux frameworks, dont PyTorch, TensorFlow, et d'autres.

Étapes :

1) **Entraîner le modèle en Python (par exemple avec TensorFlow ou PyTorch)**.
2) **Exporter le modèle vers le format ONNX**.

Pour PyTorch **:**

```
import torch.onnx
model = xxxx # ton modèle PyTorch
entree = torch.randn(x, xx, xxx, xxx)  # Forme d'entrée du modèle
torch.onnx.export(model, entree, "model.onnx")
```

Pour TensorFlow :

- il existe aussi une méthode d'exportation vers ONNX à travers des outils comme *tf2onnx*.

3) **Importer et utiliser le modèle ONNX dans Java** :

Utiliser une bibliothèque Java comme **ONNX Runtime** (https://onnxruntime.ai/) pour charger et exécuter des modèles ONNX.

Exemple d'utilisation avec ONNX Runtime en Java :

```
import ai.onnxruntime.*;

public class OnnxModel {
    public static void main(String[] args) throws Exception {
        OnnxModel model = OnnxModel.loadModel("model.onnx");
        // Charger les entrées et effectuer des prédictions
    }
}
```

3. Utiliser TensorFlow Java API

Si votre modèle est entraîné avec **TensorFlow**, vous pouvez l'exporter en format .pb (Protobuf) et utiliser l'API Java de TensorFlow pour l'exécuter directement.

Étapes :

1) **Exporter le modèle en Python** :

```
model.save("model.pb")
```

2) **Charger et exécuter ce modèle en Java avec TensorFlow Java API** :

Vous devez ajouter la dépendance tensorflow-core dans votre projet Java (Maven ou Gradle).

Exemple de code Java :

```
import org.tensorflow.Graph;
import org.tensorflow.Session;
import org.tensorflow.Tensor;

public class TensorFlowModel {
    public static void main(String[] args) {
        try (Graph graph = new Graph()) {
            // Charger le modèle
            graph.importGraphDef(Files.readAllBytes(Paths.get("model.pb")));

            try (Session session = new Session(graph)) {
                // Effectuer des prédictions
                Tensor inputTensor = Tensor.create(inputData);
                Tensor result = session.runner().feed("input",
inputTensor).fetch("output").run().get(0);
```

```
            }
        }
    }
}
```

4. Exporter le modèle en format Java natif (via un wrapper)

Si vous avez un modèle relativement simple ou si vous êtes prêt à le réécrire en Java, vous pouvez créer un wrapper ou une réimplémentation de votre modèle en Java. Ce n'est pas optimal pour des modèles complexes, mais cela reste une option.

Conclusion

La méthode la plus courante et flexible pour transférer un modèle Python vers Java est d'utiliser **ONNX**, car il permet de transférer facilement des modèles entre plusieurs frameworks et est pris en charge par des bibliothèques Java comme **ONNX Runtime**. Si vous utilisez TensorFlow ou Keras, vous pouvez utiliser des outils comme **DL4J** ou **TensorFlow Java API** pour charger les modèles.