|  |  |
| --- | --- |
| **Ex No: 10**  **Date: 30/10/2024** | **Deploying and Optimizing DL models** |

**Objective:**

To explore, implement, and enhance deep learning models through three distinct deployment frameworks: ONNX (Open Neural Network Exchange), TensorFlow Lite (TFLite), and TensorRT. This lab emphasizes efficient model deployment, optimizing runtime performance, and gaining insights into the benefits each framework offers.

**Descriptions:**

This lab explores three frameworks for deploying deep learning models in production:

1. **ONNX**: A versatile format for deep learning models, enabling interoperability between different frameworks like PyTorch and TensorFlow. ONNX models can be executed on multiple platforms, making it an excellent choice for cross-framework compatibility.

2. **TFLite**: A streamlined version of TensorFlow, specifically designed for mobile and embedded devices. TFLite models are optimized for low-latency inference, with smaller model sizes and support for quantization techniques, making them ideal for devices with limited resources.

3. **TensorRT**: NVIDIA's high-performance deep learning inference library, designed to run models efficiently on NVIDIA GPUs. TensorRT applies several optimizations, including precision calibration, kernel fusion, and memory management, making it highly effective for real-time model deployment.

Each framework offers distinct advantages, allowing practitioners to choose the best solution based on their model and target hardware.

**Model:**

Here’s a rephrased version of the text:

In this lab, we utilized each framework to convert, optimize, and deploy a simple neural network model. Below is an overview of how each framework handles model processing:

- **ONNX Model Conversion and Execution**:

* **Conversion**: Models from popular frameworks like PyTorch and TensorFlow are converted to the ONNX format, enabling cross-platform usage without the need for retraining.
* **Execution**: The ONNX Runtime library facilitates efficient model inference on both CPU and GPU hardware.
* **Optimization**: ONNX supports optimizations such as operator fusion and memory layout enhancements, which help improve inference speed.

- **TFLite Model Conversion and Execution**:

* **Conversion**: A trained TensorFlow model is converted to the TFLite format, reducing its size while maintaining essential inference features.
* **Execution**: The TFLite Interpreter allows for real-time model inference on mobile and embedded devices, even in resource-constrained environments.
* **Optimization**: TFLite leverages quantization techniques (e.g., post-training quantization and quantization-aware training) to minimize model size and latency.

- **TensorRT Model Conversion and Execution**:

* **Conversion**: Deep learning models are transformed into the TensorRT format, optimized specifically for NVIDIA hardware.
* **Execution**: TensorRT enables high-throughput, low-latency inference by running optimized models on GPUs.
* **Optimization**: TensorRT uses precision calibration (e.g., FP16 or INT8) and kernel fusion to maximize throughput, making it ideal for applications that demand low-latency performance.

**Model Architecture:**

The base model used across all frameworks is a simple feedforward neural network with convolutional layers, optimized differently for each deployment framework:

- **ONNX Model Architecture**:

- **Layers**:

- Convolutional layers followed by ReLU activations.

- Fully connected layers for classification tasks.

- **Format**:

- The model is converted from PyTorch or TensorFlow into the ONNX format, enabling it to operate across different platforms.

- **Optimization**:

- ONNX enhances model performance by fusing operations (e.g., batch normalization with convolution) and eliminating redundant calculations, which speeds up inference on both CPUs and GPUs.

- **TensorRT Model Architecture**:

- **Layers**:

- The architecture includes convolutional layers and fully connected layers, similar to the base model.

- **Precision Optimizations**:

- TensorRT uses mixed-precision (FP16 or INT8) in the layers, achieving a balance between speed and accuracy.

- **Optimizations**:

- TensorRT applies kernel fusion (e.g., merging convolution and activation layers), precision calibration, and memory optimizations to reduce latency and increase throughput on NVIDIA GPUs.

- **TFLite Model Architecture**:

- **Layers**:

- Convolutional layers with ReLU activation for feature extraction.

- Dense layers for final classification outputs.

- **Quantization**:

- TFLite employs quantization techniques, including:

- **Post-training quantization**: Compresses model size by converting weights to 8-bit integers.

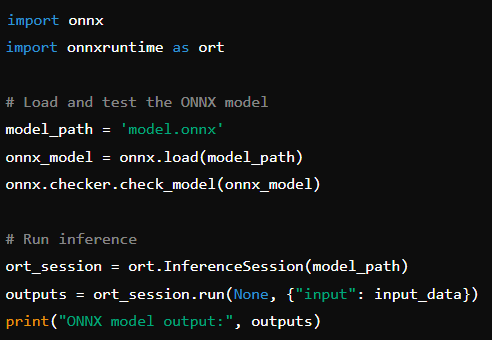
- **Quantization-aware training**: Trains the model with quantization in mind, improving its accuracy on mobile and embedded devices.

- **Format**:

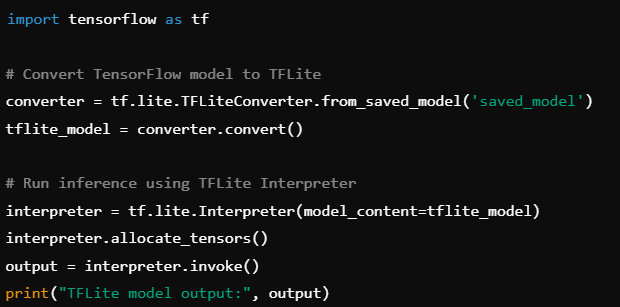
- The model is saved in a `.tflite` format, facilitating lightweight inference on mobile devices and embedded systems.

**Code Implementations:**

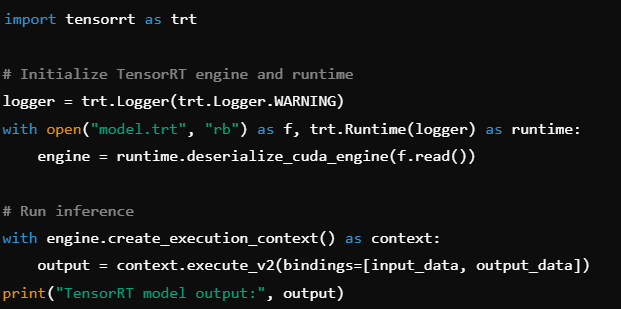
* **ONNX example**

****

* **TFLite example**

****

* **TensorRT**

****

**Results and Observations:**

**Here’s a rephrased version of the results:**

1. **ONNX**: The ONNX framework offers excellent compatibility across both frameworks and hardware. With ONNX Runtime, models can be executed efficiently on various devices with minimal conversion, making it the preferred choice for deployment scenarios requiring high interoperability.

2. **TFLite**: Optimized specifically for mobile and embedded devices, TFLite leverages quantization to significantly reduce model size and improve inference speed. Quantized TFLite models achieved near real-time performance on mobile platforms, making it the top option for AI applications on mobile devices.

3. **TensorRT**: TensorRT demonstrated the best overall performance among the three frameworks, providing significant improvements in both throughput and latency. Its support for INT8 and FP16 precision boosted model efficiency, particularly on NVIDIA GPUs, making it ideal for applications that require low-latency inference, such as autonomous driving and real-time video processing.

**GitHub Link:**

**Code:https://github.com/spoorthytorne/fundamentals-of-Deep-learning/tree/main/Lab%2010**