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

**Objective:**

To understand, implement, and optimize deep learning models using three different frameworks for model deployment: **ONNX (Open Neural Network Exchange)**, **TFLite (TensorFlow Lite)**, and **TensorRT**. This lab focuses on deploying models efficiently, improving runtime performance, and understanding the advantages of each deployment framework.

**Descriptions:**

The lab covers three frameworks for deploying deep learning models in production environments:

1. **ONNX**: A universal format for deep learning models that allows interoperability between various frameworks (e.g., PyTorch, TensorFlow). ONNX models can be executed on a variety of platforms, making it an ideal choice for cross-framework compatibility.
2. **TFLite**: A lightweight version of TensorFlow designed specifically for mobile and embedded devices. TFLite models are optimized for low-latency inference, with reduced model size and support for quantization techniques, making it suitable for resource-constrained devices.
3. **TensorRT**: A high-performance deep learning inference library developed by NVIDIA, which optimizes models to run efficiently on NVIDIA GPUs. TensorRT applies a variety of optimizations, including precision calibration, kernel fusion, and memory management, which makes it particularly effective for deploying models in real-time applications.

Each of these frameworks offers unique benefits tailored to specific deployment needs, enabling practitioners to select the optimal approach for their model and target hardware.

**Model:**

In this lab, we worked with each framework to convert, optimize, and deploy a simple neural network model. Here’s a breakdown of how each framework processes models:

* **ONNX Model Conversion and Execution**:
  + **Conversion**: Models from popular frameworks (e.g., PyTorch or TensorFlow) are converted to the ONNX format, enabling them to be used across platforms without retraining.
  + **Execution**: The ONNX Runtime library allows for efficient model inference on both CPUs and GPUs.
  + **Optimization**: ONNX supports model optimizations such as operator fusion and memory layout adjustments to improve inference speed.
* **TFLite Model Conversion and Execution**:
  + **Conversion**: A trained TensorFlow model is converted into the TFLite format, reducing model size while retaining essential features for inference.
  + **Execution**: TFLite Interpreter supports running models on mobile and embedded devices, enabling real-time inference in resource-limited environments.
  + **Optimization**: TFLite offers quantization techniques (e.g., post-training quantization and quantization-aware training) to further reduce model size and latency.
* **TensorRT Model Conversion and Execution**:
  + **Conversion**: Deep learning models are converted into the TensorRT format, which is optimized specifically for NVIDIA hardware.
  + **Execution**: TensorRT provides a high-throughput and low-latency inference by running optimized models on GPUs.
  + **Optimization**: TensorRT applies precision calibration (e.g., FP16 or INT8 precision) and kernel fusion to maximize throughput, making it ideal for latency-sensitive applications.

**Model Architecture:**

The base model architecture used for each framework consists of a simple feedforward neural network with convolutional layers, optimized differently for each framework.

**ONNX Model Architecture**

* **Layers**:
  + Convolutional layers followed by ReLU activations
  + Fully connected layers for classification
* **Format**:
  + Converted from a PyTorch or TensorFlow model into the ONNX format, enabling the model to run across different platforms.
* **Optimization**:
  + ONNX optimizes the model by fusing operations (e.g., batch normalization with convolutions) and reducing redundant calculations, resulting in improved inference time on both CPUs and GPUs.

**TensorRT Model Architecture**

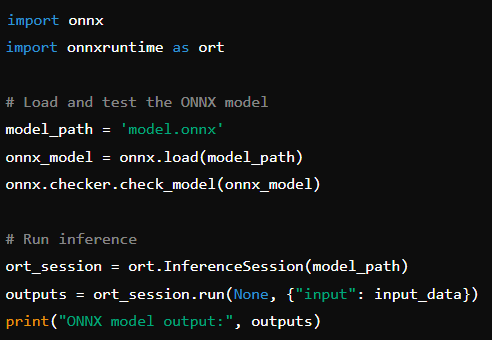
* **Layers**:
  + Similar convolutional and fully connected layers as in the base model
* **Precision Optimizations**:
  + Uses mixed-precision (FP16 or INT8) for layers, balancing speed and accuracy.
* **Optimizations**:
  + TensorRT applies kernel fusion (e.g., merging activation layers with convolutions), precision calibration, and memory optimizations to minimize latency and improve throughput on NVIDIA GPUs.

**TFLite Model Architecture**

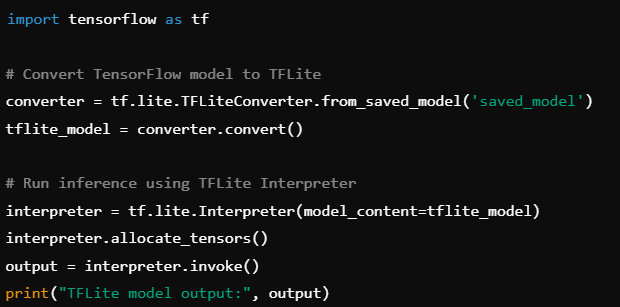
* **Layers**:
  + Convolutional layers with ReLU activation for feature extraction
  + Dense layers for final classification output
* **Quantization**:
  + The TFLite model uses quantization techniques, such as:
    - **Post-training quantization**: Reduces model size by converting weights to 8-bit integers.
    - **Quantization-aware training**: Trains the model with quantization in mind, improving model accuracy on mobile and embedded devices.
* **Format**:
  + Saved as a .tflite file, enabling lightweight inference on mobile devices and embedded systems.

**Code Implementations:**

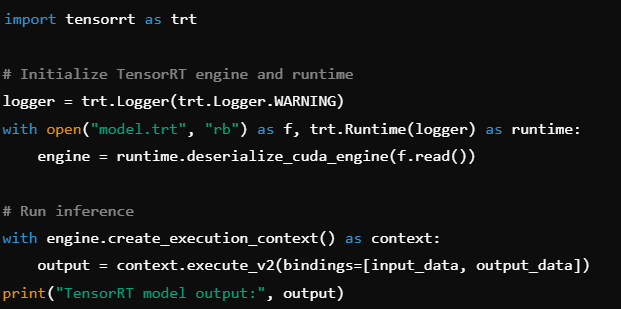
* **ONNX example**

****

* **TFLite example**

****

* **TensorRT**

****

**Results and Observations:**

1. **ONNX**: The ONNX framework provided excellent compatibility across frameworks and hardware. By using ONNX Runtime, models can run efficiently on different devices with minimal conversion, making it ideal for deployment where interoperability is a priority.
2. **TFLite**: TFLite is optimized for mobile and embedded devices, with quantization features that effectively reduce the model size and improve inference time. Quantized TFLite models achieved near real-time performance on mobile devices, making it the best choice for mobile AI applications.
3. **TensorRT**: TensorRT delivered the highest performance among the three frameworks, with substantial improvements in throughput and latency. Its support for INT8 and FP16 precision further enhanced model performance, particularly on NVIDIA GPUs, making it highly suitable for applications demanding low-latency inference, such as autonomous driving or real-time video processing.

**GitHub Link:**

**Code:**