Lab 04: Deploying Models on Edge Devices

Omar Diaa Eldin
M.Eng. Artificial Intelligence for Smart Sensors & Actuators
Prof. Tobias Schaffer

June 13, 2025

1 Introduction

This lab focused on implementing, training, evaluating, and exporting a simple neural network using both TensorFlow and PyTorch. The main goal was to understand their differences in workflow, performance, and model exportability, and how to deploy trained models (TFLite and ONNX) on a Raspberry Pi and run inference using dummy data.

2 Model Architecture and Dataset

- Dataset: MNIST handwritten digits (28x28 grayscale)
- Layers:
  - Flatten layer
  - Dense layer with 64 ReLU units
  - Output layer with 10 classes

3 Implementation

3.1 Development Environment for Models

- **CPU:** Intel(R) Core(TM) i7-7700HQ @ 2.80GHz (4 cores, 8 threads)
- **GPU:** NVIDIA GeForce GTX 1070, 8 GB VRAM
- **RAM:** 16 GB DDR4
- **OS:** Windows 10 64-bit
- **Programming Language:** Python
- **Python Version:** Python 3.10
- **IDE:** PyCharm

Development Environment for Models on Edge Device

- **Device:** Raspberry Pi 5
- **RAM:** 8 GB LPDDR4
- **OS:** Raspberry Pi OS 64-bit
- **Python Version:** Python 3.10

3.2 Code Repository

The full source code for this project is available on GitHub at: https://github.com/omardmf/programming-lecture

3.3 TensorFlow

```
import tensorflow as tf
import time

model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
   metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5)
```

3.4 PyTorch

```
import torch
import torch.nn as nn

class MNISTModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(784, 64)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(64, 10)

    def forward(self, x):
        x = x.view(-1, 784)
        return self.fc2(self.relu(self.fc1(x)))
```

4 Metrics

4.1 TensorFlow

```
print("Training␣started...")
start_time = time.time()
model.fit(x_train, y_train, epochs=5, batch_size=64, verbose=2)
training_time = time.time() - start_time

test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
```

4.2 PyTorch

```
model.eval()
correct = 0
total = 0

start_infer = time.time()
with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
inference_time = time.time() - start_infer

accuracy = correct / total
```

5 Results

Framework	**Accuracy (%)**	**Inference Time (ms)**
TensorFlow	96.74	1.746
PyTorch	95.67	2726.5

Table 1: Performance Comparison of Models

6 Model Export

- TensorFlow Lite:

```
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()
```

- PyTorch to ONNX:

```
dummy_input = torch.randn(1, 784)
torch.onnx.export(model, dummy_input, "model.onnx")
```

7 Deployment Steps on Raspberry Pi

1. Prepare Raspberry Pi with a virtual environment:

```
sudo apt update
sudo apt install python3-full python3-venv
python3 -m venv ~/env1
source ~/env1/bin/activate
pip install numpy tensorflow onnxruntime
```

2. Transfer models via SCP:

```
scp model.tflite pi@raspberrypi:~/
scp model.onnx pi@raspberrypi:~/
```

3. Run TensorFlow Lite inference:

```
interpreter = tf.lite.Interpreter(model_path="model.tflite")
interpreter.allocate_tensors()
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
interpreter.set_tensor(input_details[0]['index'], test_image)
interpreter.invoke()
output = interpreter.get_tensor(output_details[0]['index'])
```

4. Run ONNX inference:

```
import onnxruntime as ort
session = ort.InferenceSession("model.onnx")
outputs = session.run(None, {"input": test_image})
```

8 Results

Framework	Inference Time (ms)	Predicted Class
TFLite	1.746	3
ONNX	2.995	4

Table 2: Performance Comparison of Models on Edge Device

9 Challenges, Limitations, and Error Analysis

9.1 Error Analysis

- Model mismatch errors during ONNX export due to incorrect input shape definitions.
- Slow inference or segmentation faults on the Raspberry Pi when attempting to run unoptimized models or incompatible TensorFlow versions.

9.2 Limitations of the Implementation

- The ONNX model predicted a different class than the TensorFlow Lite model for the same dummy input, indicating possible numerical differences in backend operations.
- Limited quantization and compression were applied; more optimized versions of the models would yield better performance on edge devices.

10 Discussion

The lab successfully demonstrated end-to-end deployment of deep learning models on edge devices. Training and inference were faster on TensorFlow, particularly due to its built-in support for TFLite conversion. PyTorch offered more flexibility during model construction, but ONNX conversion required more effort and had higher inference time. Interestingly, the predicted classes for a sample image differed slightly: TensorFlow Lite predicted class `3`, while ONNX predicted class `4`. This highlights that different runtimes may produce minor numerical variations, especially with uncalibrated floating point weights.
In terms of deployment:

- TFLite was easier to integrate with the Pi and showed faster performance.
- ONNX required more setup but demonstrated good compatibility once configured.

11 Conclusion

This lab compared TensorFlow and PyTorch in terms of model development, evaluation, and deployment to edge hardware. TensorFlow provided a smoother experience from training to deployment via TFLite. PyTorch required additional steps but benefited from clearer modularity.

12 References

- TensorFlow documentation: `https://www.tensorflow.org/`
- PyTorch documentation: `https://pytorch.org/`
- ONNX Runtime: `https://onnxruntime.ai/`
- MNIST Dataset: `http://yann.lecun.com/exdb/mnist/`