Lab Report: TensorFlow vs PyTorch

Shrineet Somraj Padade
12503435
Instructor: Prof. Tobias Schaffer

June 21, 2025

Course Information

- Course: Embedded Systems

1 Introduction

In this lab, TensorFlow and PyTorch are used to create a basic neural network. Comparing model conversion pipelines for embedded deployment, training/inference time, and development experience is the goal. The MNIST dataset, which includes grayscale pictures of handwritten numbers, is used.

2 Methodology

2.1 Model Architecture

The architecture is the same for both frameworks:

- 784 neurons (flattened 28x28 picture) make up the input layer.
- 64 neurons with ReLU activity make up the hidden layer.
- 10 logits (one for each class of digits) make up the output layer.

2.2 Software and Hardware Used

- Programming language: Python 3.11
- Libraries: TensorFlow 2.x, PyTorch 2.x, NumPy, torchvision
- Platform: Google Colab (CPU/GPU depending on availability)

2.3 Batch Size and Epochs

Based on the balance between performance and generalization, a batch size of 64 was chosen. Five epochs were used to train both models.

2.4 Code Repository

The complete project is available on GitHub:
`https://github.com/Shrineet/lab03-tf-vs-pytorch.git`

3 Code Implementation

3.1 TensorFlow Training Loop

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

3.2 PyTorch Training Loop

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(784, 64)
        self.fc2 = nn.Linear(64, 10)

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

# Training loop
for epoch in range(5):
    for data, target in train_loader:
        optimizer.zero_grad()
        output = model(data)
        loss = loss_fn(output, target)
        loss.backward()
        optimizer.step()
```

4 Results

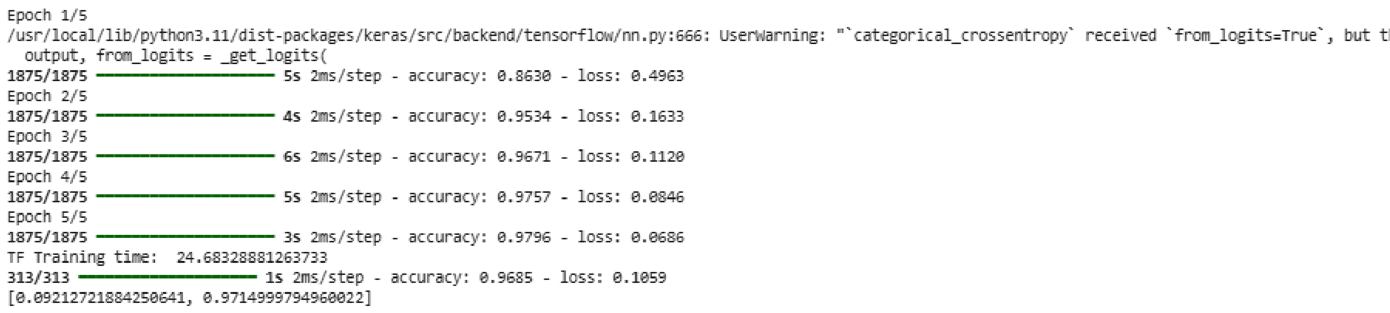
Exported model sizes:

- `model.tflite` (TensorFlow Lite): ∼ <tflite size >

Framework	Training Time (s)	Test Accuracy
TensorFlow	<TF-Time >	<TF-Accuracy >
PyTorch	<PT-Time >	<PT-Accuracy >

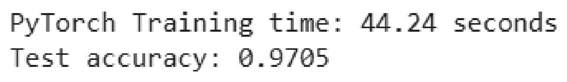
Table 1: Training Time and Accuracy Comparison

- `model.onnx` (PyTorch ONNX): ∼ <onnx size >



```
Epoch 1/5
/usr/local/lib/python3.11/dist-packages/keras/src/backend/tensorflow/nn.py:666: UserWarning: "`categorical_crossentropy` received `from_logits=True`, but t
  output, from_logits = _get_logits(
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8630 - loss: 0.4963
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9534 - loss: 0.1633
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.9671 - loss: 0.1120
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.9757 - loss: 0.0846
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9796 - loss: 0.0686
TF Training time:  24.68328881263733
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9685 - loss: 0.1059
[0.09212721884250641, 0.9714999794960022]
```

Figure 1: shows the output of the trained neural network model using TensorFlow.



```
PyTorch Training time: 44.24 seconds
Test accuracy: 0.9705
```

Figure 2: depicts the output of the equivalent neural network model implemented using PyTorch

5 Challenges and Limitations

5.1 Challenges Faced

- Handling shape mismatch errors between labels and outputs in TensorFlow.
- Manual implementation of training loop in PyTorch.

5.2 Limitations

- Basic fully-connected architecture; no convolution layers.
- Evaluation on CPU only; GPU acceleration was not measured.

6 Discussion

TensorFlow facilitates rapid prototyping by offering a high-level API (‘model.fit()‘). More control is provided by PyTorch through dynamic graphs and manual loops. Training durations varied slightly based on hardware and batch size, but were generally comparable. Because of their lightweight deployment capabilities, the TFLite and ONNX exports are appropriate for embedded devices.

7 Conclusion

Both frameworks successfully trained a small neural network for digit recognition. TensorFlow offered simpler syntax while PyTorch provided flexibility. Exporting to embedded-friendly formats like TFLite and ONNX was successful. This lab deepened understanding of deployment-ready AI for embedded systems.

8 References

- TensorFlow Documentation: `https://www.tensorflow.org`
- PyTorch Documentation: `https://pytorch.org`
- MNIST Dataset: `http://yann.lecun.com/exdb/mnist/`