Lab Report: **Tensorflow and PyTorch**

Name: Snehalatha Meesala
Student ID: 12503817
Course Name: Artificial Intelligence for smart sensors and Actuators
Instructor: Prof. Tobias Schaffer

Date: 11-07-2025

1 Introduction

The objective of this lab was to implement and compare a simple neural network model using two popular deep learning frameworks: TensorFlow and PyTorch. The task involved training the same architecture on the MNIST dataset, measuring training/inference performance, and exporting the models to lightweight formats (TFLite and ONNX) for deployment

2 Methodology

A feedforward neural network was trained on the MNIST dataset, consisting of 28×28 grayscale images of handwritten digits. The network architecture includes:- Flattened input (784 features)- Dense layer with 64 ReLU units- Output layer with 10 units The network was implemented in both TensorFlow and PyTorch with equivalent logic to ensure a fair comparison

2.1 Software and Hardware Used

- Programming language: Python3
- Libraries: TensorFlow 2.x, PyTorch, torchvision, NumPy

2.2 Code Repository

The full source code for this project is available on GitHub at:

`https://github.com/Snehalathameesala/Tensorflow-and-PyTorch`

2.3 Code Implementation

PyTorch Model:

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(784, 64)  # Fully connected layer (input ->
            hidden)
        self.fc2 = nn.Linear(64, 10)   # Fully connected layer (hidden
           -> output)

    def forward(self, x):
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        x = F.softmax(x, dim=1)  # Apply softmax along class dimension
        return x
```

TensorFlow Model:

```
import tensorflow as tf
from tensorflow.keras import layers

model = tf.keras.Sequential([
    layers.Input(shape=(784,)),        # Input layer for flattened 28x28
        image
    layers.Dense(64, activation='relu'),  # Hidden layer with ReLU
    layers.Dense(10, activation='softmax')  # Output layer for 10
       classes
])
```

3 Results

Framework	Training Time	Test Accuracy	Export Format
TensorFlow	33 seconds	96.73	TFLite
PyTorch	63 seconds	97.41	ONNX

Table 1: Result of Pytorch and TensorFlow

4 Challenges, Limitations, and Error Analysis

4.1 Challenges Faced

- Aligning the architecture and hyperparameters identically across both frameworks.
- Ensuring model export procedures worked correctly in Colab.

4.2 Error Analysis

Some common errors that occurred during the development:

- Initial model export failed in PyTorch due to incorrect input shape.
- Minor syntax differences caused temporary runtime issues

4.3 Limitations of the Implementation

- No early stopping or learning rate adjustments were applied.
- Dataset was normalized but not augmented.- Evaluation was based only on accuracy

5 Discussion

Both frameworks produced comparable results, but PyTorch demonstrated slightly better test accuracy at the cost of longer training time. TensorFlow's Keras API was more concise and easier to use for quick prototyping, while PyTorch offered more flexibility and transparency.

6 Conclusion

The lab successfully demonstrated the implementation and evaluation of identical neural network models in TensorFlow and PyTorch. While TensorFlow provided faster development and easier deployment, PyTorch yielded marginally better accuracy

7 References

- TensorFlow Lite Guide: `https://www.tensorflow.org/lite`
- Sense HAT API Documentation: `https://pythonhosted.org/sense-hat/`