

Part 1: Theoretical Understanding (40%)

1. Short Answer Questions

Q1: Explain the primary differences between TensorFlow and PyTorch. When would you choose one over the other?

TensorFlow is a developer friendly library or framework that focuses on production and scalability while PyTorch is a framework that researchers who experiment new ideas use due to its flexibility.

TensorFlow for deployment (mobile, web) and large-scale stable applications. PyTorch for research in academia, prototyping and for complex dynamic models.

Q2: Describe two use cases for Jupyter Notebooks in AI development.

Exploratory Data Analysis -Visualizing and cleaning data instantly

Rapid Model Prototyping- Building, training, checking and tweaking models quickly without re-running long scripts.

Q3: How does spaCy enhance NLP tasks compared to basic Python string operations?

Basic Python string operations treat text as characters, while spaCy treats text as language. For example, A string operation sees "A-p-p-l-e," but spaCy sees "Company Name or an organization."

2. Comparative Analysis

Compare Scikit-learn and TensorFlow in terms of:

- Target applications (e.g., classical ML vs. deep learning).

Scikit-learn -Classical Machine Learning (Classification, Regression, Clustering) on structured/tabular data for example spam filters and credit scoring.

TensorFlow- Used for Deep Learning (Neural Networks) for unstructured data for example image recognition, NLP and reinforcement learning

- Ease of use for beginners.

Scikit-learn – Very simple to use as it provides a clean, consistent API and is great for quickly testing standard algorithms due to its non-deep-learning focus.

TensorFlow- Was initially more complex, but it's integration of the Keras API makes building and training complex deep learning models very user-friendly and accessible for beginners.

- Community support.

Scikit-learn – Widely used in academia and traditional Machine Learning and it's known for reliable, well-documented algorithms.

TensorFlow- It powers google, YouTube. Widely used by industries, backed by Google. It focuses on deep learning and getting complex models into real-world products.

Screenshots of model outputs (e.g., accuracy graphs, NER results)

Task 1: Iris Classification (Scikit-learn)

- Model evaluation output: The printed accuracy, precision, recall, and classification report.
- Decision tree plot: The visual tree generated using plot_tree().

The screenshot shows a Jupyter Notebook interface with the following content:

Classification Report:

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	19
versicolor	1.00	1.00	1.00	13
virginica	1.00	1.00	1.00	13
accuracy			1.00	45
macro avg	1.00	1.00	1.00	45
weighted avg	1.00	1.00	1.00	45

Decision Tree Visualization for Iris Classification

```

graph TD
    Root["petal length (cm) <= 2.45  
gini = 0.664  
samples = 105  
value = [31, 37, 37]  
class = versicolor"] --> L1_1["petal length (cm) <= 4.75  
gini = 0.5  
samples = 74  
value = [0, 37, 37]  
class = versicolor"]
    Root --> L1_2["petal width (cm) <= 1.75  
gini = 0.214  
samples = 41  
value = [0, 5, 36]  
class = virginica"]
    L1_1 --> L2_1["petal width (cm) <= 1.6  
gini = 0.059  
samples = 33  
value = [0, 32, 1]  
class = versicolor"]
    L1_1 --> L2_2["petal length (cm) <= 4.95  
gini = 0.5  
samples = 8  
value = [0, 4, 4]  
class = versicolor"]
    L1_2 --> L2_3["petal width (cm) <= 1.75  
gini = 0.214  
samples = 41  
value = [0, 5, 36]  
class = virginica"]
    L2_1 --> L3_1["petal width (cm) <= 1.55  
gini = 0.444  
samples = 6  
value = [0, 2, 4]  
class = virginica"]
    L2_1 --> L3_2["sepal width (cm) <= 3.1  
gini = 0.444  
samples = 3  
value = [0, 1, 2]  
class = virginica"]
    L2_2 --> L3_3["petal length (cm) <= 5.45  
gini = 0.444  
samples = 3  
value = [0, 2, 1]  
class = versicolor"]
    L2_2 --> L3_4["gini = 0.0  
samples = 2  
value = [0, 2, 0]  
class = versicolor"]
    L3_1 --> L4_1["gini = 0.0  
samples = 3  
value = [0, 0, 3]  
class = virginica"]
    L3_1 --> L4_2["gini = 0.0  
samples = 2  
value = [0, 2, 0]  
class = versicolor"]
    L3_2 --> L4_3["gini = 0.0  
samples = 1  
value = [0, 0, 1]  
class = virginica"]
    L3_3 --> L4_4["gini = 0.0  
samples = 2  
value = [0, 0, 2]  
class = virginica"]
    L3_3 --> L4_5["gini = 0.0  
samples = 1  
value = [0, 1, 0]  
class = virginica"]
  
```

Task 2: MNIST CNN (TensorFlow)

- Test accuracy printout-The final model.evaluate() result.
- Prediction visualization- plot showing 5 sample digits with true vs predicted labels.

Task2DL-MNISTdataset.ipynb

```

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434    0s 0us/step
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass an `input_shape`/'input_dim' super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/5
844/844    47s 53ms/step - accuracy: 0.8732 - loss: 0.4328 - val_accuracy: 0.9812 - val_loss: 0.0644
Epoch 2/5
844/844    48s 56ms/step - accuracy: 0.9821 - loss: 0.0593 - val_accuracy: 0.9852 - val_loss: 0.0499
Epoch 3/5
844/844    45s 54ms/step - accuracy: 0.9878 - loss: 0.0400 - val_accuracy: 0.9865 - val_loss: 0.0467
Epoch 4/5
844/844    45s 54ms/step - accuracy: 0.9911 - loss: 0.0297 - val_accuracy: 0.9868 - val_loss: 0.0479
Epoch 5/5
844/844    44s 52ms/step - accuracy: 0.9931 - loss: 0.0225 - val_accuracy: 0.9857 - val_loss: 0.0504
313/313    3s 10ms/step - accuracy: 0.9821 - loss: 0.0516
  ✓ Test Accuracy: 0.9852 | Test Loss: 0.0439
1/1    0s 129ms/step

```

Task2DL-MNISTdataset.ipynb

```

  ✓ Test Accuracy: 0.9825 | Test Loss: 0.0524
313/313    3s 9ms/step
Digit 0 Accuracy: 0.9939
Digit 1 Accuracy: 0.9947
Digit 2 Accuracy: 0.9767
Digit 3 Accuracy: 0.9980
Digit 4 Accuracy: 0.9562
Digit 5 Accuracy: 0.9753
Digit 6 Accuracy: 0.9875
Digit 7 Accuracy: 0.9903
Digit 8 Accuracy: 0.9641
Digit 9 Accuracy: 0.9851
1/1    0s 59ms/step
True: 6 Pred: 6
True: 2 Pred: 2
True: 3 Pred: 3
True: 7 Pred: 7
True: 2 Pred: 2

```

Variables Terminal ✓ 19:55 Python 3

Task 3: NLP with spaCy

- NER output: Printed named entities like ('Apple iPhone 13', 'PRODUCT') and sentiment result: Printed sentiment label for each review (e.g., "Positive", "Negative").

Task3NLP_spacy.ipynb

```

Review: I love my new Apple iPhone 13! The camera is amazing.
Entities: [('Apple', 'ORG')]
Sentiment: Positive

Review: Terrible experience with the Samsung Galaxy. It kept freezing.
Entities: []
Sentiment: Negative

Review: The Sony headphones are comfortable and sound great.
Entities: [('Sony', 'ORG')]
Sentiment: Positive

Review: Avoid this Lenovo laptop. Battery life is awful.
Entities: [('Lenovo', 'ORG')]
Sentiment: Negative

Review: My new Nike running shoes are super comfy and stylish!
Entities: [('Nike', 'ORG')]
Sentiment: Positive

```

Ethic Considerations Fairness Check: Accuracy MNIST dataset

The screenshot shows a Jupyter Notebook interface with the following details:

- Title:** Task2DL-MNISTdataset.ipynb
- Toolbar:** File, Edit, View, Insert, Runtime, Tools, Help
- Search Bar:** Q Commands, + Code, + Text, Run all
- Runtime Status:** RAM, Disk
- Output Cell:** Displays test accuracy and loss information:
 - Test Accuracy: 0.9825 | Test Loss: 0.0524
 - 313/313 3s 9ms/step
 - Digit 0 Accuracy: 0.9939
 - Digit 1 Accuracy: 0.9947
 - Digit 2 Accuracy: 0.9767
 - Digit 3 Accuracy: 0.9980
 - Digit 4 Accuracy: 0.9562
 - Digit 5 Accuracy: 0.9753
 - Digit 6 Accuracy: 0.9875
 - Digit 7 Accuracy: 0.9903
 - Digit 8 Accuracy: 0.9641
 - Digit 9 Accuracy: 0.9851
- Bottom Cell:** Shows 1/1 0s 59ms/step