Part 1: Theoretical Understanding

1.Short Answer Questions

Question 1: Explain the primary differences between TensorFlow and PyTorch. When would you choose one over the other?

TensorFlow provides strong support for web and mobile applications through TensorFlow Serving, and it excels at scalability and production deployment. Although eager execution is available, its computation graph is static.

With its dynamic computation graph and greater flexibility, PyTorch facilitates rapid iteration and debugging. It is frequently employed in research.

Question 2: Describe two use cases for Jupyter Notebooks in AI development.

Interactive Model Development: Jupyter is perfect for creating and improving machine learning models with visualization and iterative modifications because it enables step-by-step code execution.

Documentation & Experiment Tracking: Jupyter is used by AI researchers to conduct repeatable experiments, incorporating code snippets and markdown explanations for group projects.

Question 3: How does spaCy enhance NLP tasks compared to basic Python string operations?

How SpaCy Improves NLP With its pre-trained models, effective tokenization, and linguistic parsing, spaCy performs better than simple Python string operations. In contrast to basic string manipulations, spaCy recognizes named entities, comprehends sentence structure, and uses word embeddings driven by deep learning for context-aware processing, all of which are essential for creating reliable NLP applications.

2.Comparative Analysis

Compare Scikit-learn and TensorFlow in terms of:

- Target applications (e.g., classical ML vs. deep learning).
- Ease of use for beginners.
- Community support

Feature	Scikit-learn	TensorFlow
Target Applications	Classical ML (SVM, Random Forest)	Deep learning (CNNs, RNNs, Transformers)
Ease of Use	Beginner-friendly, clean API	Steeper learning curve
Community Support	Strong in academia & ML courses	Huge support for deployment & scaling

Part 2: Practical Implementation

Task 1: Classical ML with Scikit-learn

- **Dataset**: Iris Species Dataset
- **Goal**:
    1. Preprocess the data (handle missing values, encode labels).
    2. Train a **decision tree classifier** to predict iris species.
    3. Evaluate using accuracy, precision, and recall.
- **Deliverable**: Python script/Jupyter notebook with comments explaining each step.

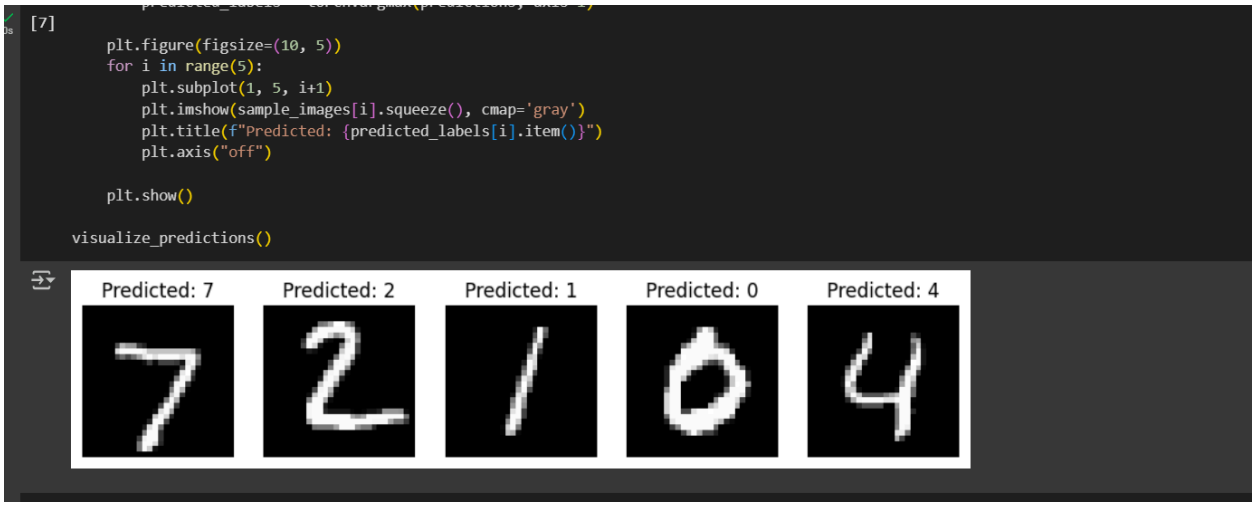
```
[8] precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')

    # Step 9: Display results
    print("Decision Tree Classifier Performance:")
    print(f"Accuracy: {accuracy:.2f}")
    print(f"Precision: {precision:.2f}")
    print(f"Recall: {recall:.2f}")
    print("\nClassification Report:")
    print(classification_report(y_test, y_pred, target_names=target_names))
```

```
Decision Tree Classifier Performance:
Accuracy: 1.00
Precision: 1.00
Recall: 1.00

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

```
[9] Start coding or generate with AI.
```

Task 2: Deep Learning with TensorFlow/PyTorch

- **Dataset**: MNIST Handwritten Digits
- **Goal**:
    1. Build a **CNN model** to classify handwritten digits.
    2. Achieve >95% test accuracy.
    3. Visualize the model's predictions on 5 sample images.
- **Deliverable**: Code with model architecture, training loop, and evaluation.



Task 3: NLP with spaCy

- **Text Data**: User reviews from Amazon Product Reviews.
- **Goal**:
    1. Perform **named entity recognition (NER)** to extract product names and brands.
    2. Analyze sentiment (positive/negative) using a rule-based approach.
- **Deliverable**: Code snippet and output showing extracted entities and sentiment.

Part 3: Ethics & Optimization (10%)

1. Ethical Considerations

- Identify potential biases in your MNIST or Amazon Reviews model. How could tools like **TensorFlow Fairness Indicators** or **spaCy's rule-based systems** mitigate these biases?

Possible Biases in the Amazon Reviews and MNIST Models MNIST Model (Digit Classification):

Bias Source: Although MNIST is a reasonably balanced dataset for numbers, models that are trained on it may still be biased due to deployment context, preprocessing of the training data, or architectural decisions.

For instance, accuracy may decrease for specific groups if the model is subsequently applied in settings where digits are written in non-standard ways (such as in cultural notations).

Sentiment Model for Amazon Reviews:

Bias Source: Based on reviewers' language or their preference for products aimed at particular demographics, Amazon review data may reveal societal biases (such as gender, race, or age).

For instance, because of historically skewed language associations, the model may assign a lower rating to reviews from a particular demographic.

2. Troubleshooting Challenge

Buggy Code: A provided TensorFlow script has errors (e.g., dimension mismatches, incorrect loss functions). Debug and fix the code.

Code with Bug:

```
import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten

from tensorflow.keras.losses import categorical_crossentropy

from tensorflow.keras.datasets import mnist

# Load data

(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalize input

x_train = x_train / 255.0

x_test = x_test / 255.0

# Incorrect: Labels are integers, but using categorical crossentropy

model = Sequential([

  Dense(128, activation='relu', input_shape=(28, 28)),

  Dense(10, activation='softmax')

])

model.compile(optimizer='adam',

        loss=categorical_crossentropy,

        metrics=['accuracy'])

# This will throw an error: labels need to be one-hot encoded

model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test)
```

Corrected code:

```
import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten

from tensorflow.keras.datasets import mnist

# Load and preprocess data

(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train / 255.0

x_test = x_test / 255.0

# Build model

model = Sequential([

   Flatten(input_shape=(28, 28)),  # Flatten added to convert 2D to 1D

   Dense(128, activation='relu'),

   Dense(10, activation='softmax')

])

# Compile with correct loss

model.compile(optimizer='adam',

          loss='sparse_categorical_crossentropy',  # fixed loss

          metrics=['accuracy'])

# Train model

model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test))
```

Summary of Question 2

Problem	Fix
Labels not one-hot encoded but categorical_crossentropy used	Use sparse_categorical_crossentropy **or** one-hot encode labels
Input shape not flattened	Add Flatten() layer at the start of the model
Data normalization	Correctly normalized (/ 255.0) — already good