Q1(a)(i) *Differences between TensorFlow and PyTorch*

Community and Ecosystem

While *TensorFlow* has a strong ecosystem backed by Google, *PyTorch* is loved by the research community, is widely used in academia and cutting-edge AI research, and is backed by Meta.

Deployment

PyTorch initially focused on research, but with TorchServe and ONNX support, deployment has become much easier while *TensorFlow* is better suited for production and deployment, especially with TensorFlow Serving, TensorFlow Lite and TensorFlow.js.

Computation Graphs

PyTorch uses dynamic computation graphs (define-and-run), you create the graph on the fly as operations are executed while *TensorFlow* uses static computation graphs (define-before-run), you define the model graph first then run it.

Debugging

While *PyTorch* is easier to debug, intuitive and flexible – making it ideal for research *TensorFlow* is less flexible for debugging and experimentation.

Q1(a)(i)-(b) *When to choose TensorFlow and PyTorch*

- When focusing on **deployment and scalability**, go with *TensorFlow*.
- When the focus is on **research, flexibility, debugging ease, and fast experimentation**, choose *PyTorch*.

Q1(a)(ii) *Use cases for Jupyter Notebooks in AI Development*

Prototyping and Testing AI Models

With Jupyter Notebooks, developers can quickly build, test and iterate on models as it is:

- Ideal for experimentation with different neural hyperparameters
- Combine code, outputs and documentation in one place for reproducibility.

Data Exploration and Processing

Data Scientists use Jupyter Notebooks to load, explore, and clean data before training Ai Models.

With Jupyter Notebooks one can:

- Visualize data using libraries like Seaborn or Matplotlib.
- Perform data wrangling, feature selection and do analysis step by step.
- Tweak code as needed to get the desired output.

Q1(a)(iii) *How spaCy enhances NLP tasks compared to basic Python string operations.*

Advanced NLP Features.

Unlike basic Python string operations, spaCy has advanced Natural Language Processing features that allow:

- Tokenization – While basic Python allows manual splitting spaCy has smart tokenization that handles abbreviations, punctuations, etc.
- Lemmatization – Converts words to their base.
- Named Entity Recognition (NER) – Detects people, dates, locations, etc.
- Dependency Parsing – Understands grammatical relationships.
- Part of Speech Tagging – Identifies verbs, adjectives, nouns, etc.

Linguistic Understanding

Basic Python works at the character or word level, but it does not understand language structure while spaCy performs linguistic analysis, recognizing parts of speech, the syntax and entities.

Example:

"A tasty burger."

Basic python will analyze it as *"A ", "tasty ", "burger. "(*no grammar or meaning while spaCy will identify "*A*" as article a, "*tasty*" as adjective, "*burger*" as a noun and their grammatical relationship.

Speed and Efficiency

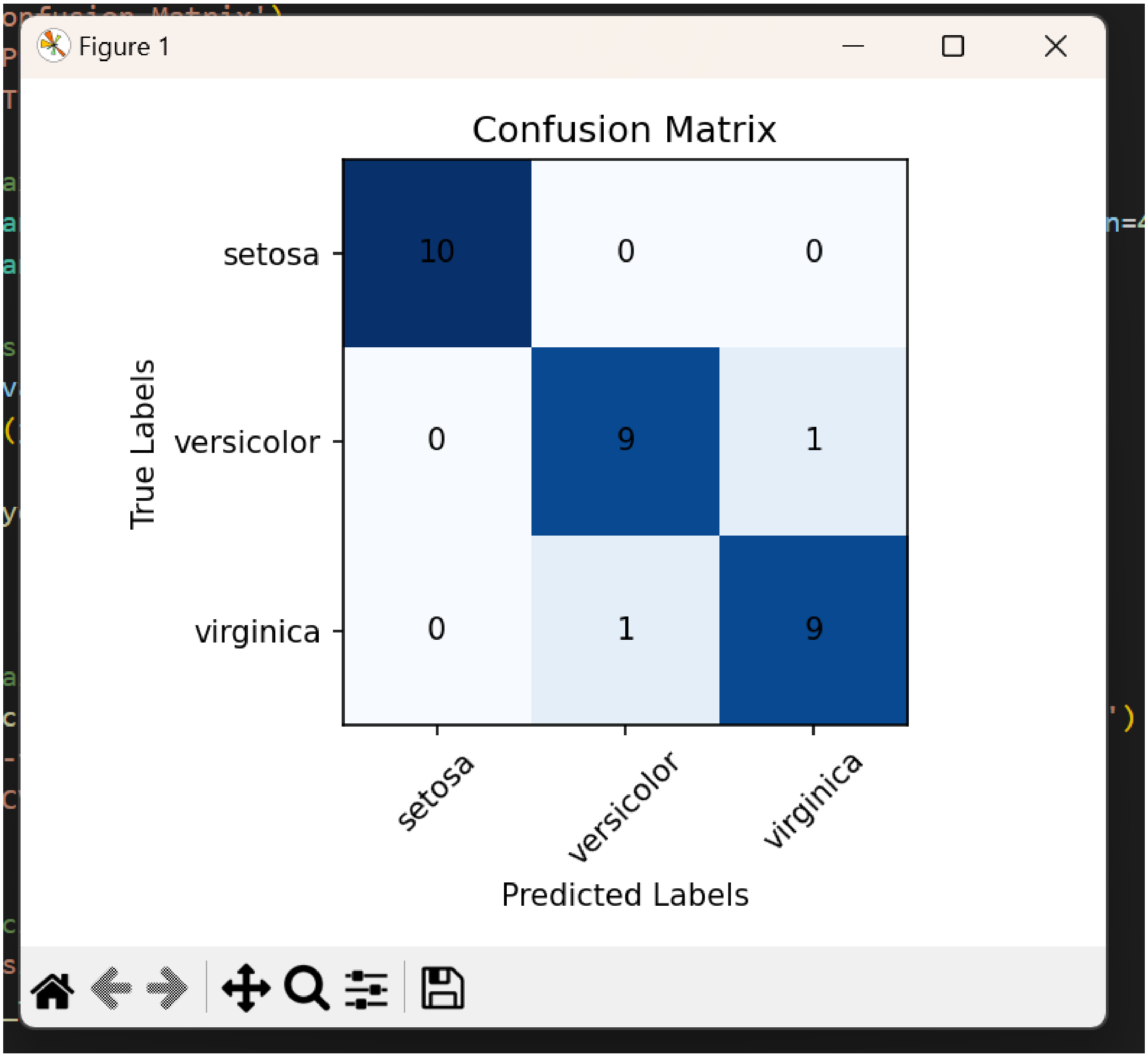
While in Python string methods one would have to manually loop over text, spaCy is highly optimized in Cython, making it faster and more memory efficient.

Q1(b) *In terms of Target applications, Ease of use for beginners and community support, compare Scikit-learn and TensorFlow.*

Indicator	**Scikit-learn**	**TensorFlow**
Target Applications	Designed for ***classical machine learning*** algos like linear regression, random forests, clustering, decision trees, and SVMS.	Built for ***deep learning and neural networks***, supporting large scale models like RNNs, CNNs and Transformers.
Ease of Use for Beginners	***Very beginner friendly***. It has simple, consistent API and models can be trained in just a few lines of code.	***Moderate learning curve*** - more complex concepts (tensors, graphs, sessions), though Keras (now part of TensorFlow) simplifies it significantly.
Community Support	It has a ***large and active data science community*** making it excellent for documentation and tutorials for traditional machine learning.	***Massive global community*** backed by Google therefore making it an extensive ecosystem for deep learning, deployment, and production use.

Q2(a)

Classical ML with Scikit-learn visualization output:



Q2(b)

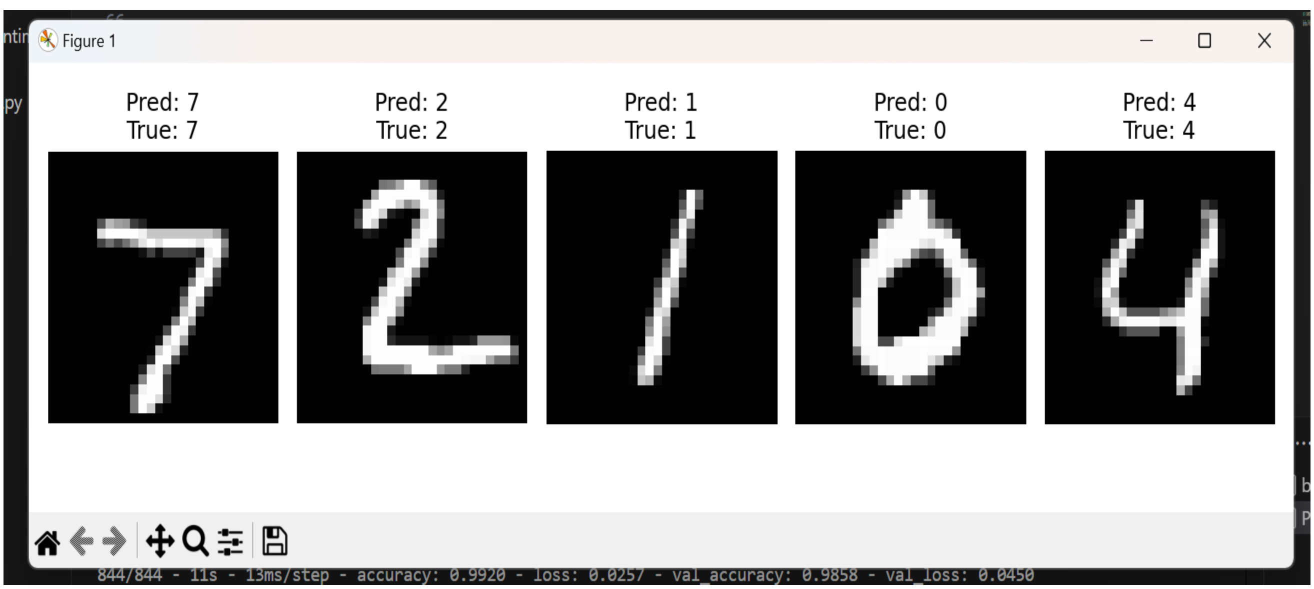
Training CNN model accuracy results:

```
Training the CNN model...

Epoch 1/5
844/844 - 10s - 12ms/step - accuracy: 0.9414 - loss: 0.1902 - val_accuracy: 0.9862 - val_loss: 0.0478
Epoch 2/5
844/844 - 9s - 11ms/step - accuracy: 0.9821 - loss: 0.0581 - val_accuracy: 0.9855 - val_loss: 0.0453
Epoch 3/5
844/844 - 11s - 13ms/step - accuracy: 0.9877 - loss: 0.0400 - val_accuracy: 0.9905 - val_loss: 0.0330
Epoch 4/5
844/844 - 11s - 13ms/step - accuracy: 0.9904 - loss: 0.0315 - val_accuracy: 0.9883 - val_loss: 0.0402
Epoch 5/5
844/844 - 11s - 13ms/step - accuracy: 0.9920 - loss: 0.0257 - val_accuracy: 0.9858 - val_loss: 0.0450

✅ Test Accuracy: 98.80%
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 115ms/step
```

Deep Learning with TensorFlow output:



Q2(c)

NLP with spaCy output:

```
40    # 4. Process each review

PROBLEMS 1    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

/Task3/nlp_spacy_sentiment.py
📝 Review: I absolutely love my new Apple iPhone! The camera quality is amazing.
🔹 Named Entities: [('Apple', 'ORG')]
💬 Sentiment: Positive 😊
------------------------------------------------------------
📝 Review: The Samsung Galaxy screen cracked within a week. Totally disappointed.
🔹 Named Entities: [('Samsung Galaxy', 'ORG'), ('a week', 'DATE')]
💬 Sentiment: Negative 😞
------------------------------------------------------------
📝 Review: This Logitech mouse is super smooth and comfortable to use.
🔹 Named Entities: [('Logitech', 'ORG')]
💬 Sentiment: Positive 😊
------------------------------------------------------------
📝 Review: I hate the battery life on my Dell laptop. It barely lasts 2 hours!
🔹 Named Entities: [('Dell', 'ORG'), ('2 hours', 'TIME')]
💬 Sentiment: Negative 😞
------------------------------------------------------------
📝 Review: The Sony headphones have great sound but the build quality feels cheap.
🔹 Named Entities: [('Sony', 'ORG')]
💬 Sentiment: Neutral 😐
------------------------------------------------------------
```

Q3(a) *Potential biases in MNIST or Amazon Review model and how TensorFlow or SpaCy can mitigate these biases.*

Ethical Considerations

Potential Biases.

Amazon Reviews (spaCy NLP)

- *Bias Source*: Rule-based sentiment models depend on specific keywording.

Example

“hate” = negative, “love” = positive.

- *Risk:* This may misclassify sarcasm, gendered languages, or cultural phrases.
- *Impact:* This could, in addition, unfairly flag or recommend products if reviews use mixed sentiment or dialect-specific language.

Mitigation Tool

spaCy Rule-based Systems

One can use custom lexicons, dependency rules, or contextual filters to avoid one-size-fits-all sentiment scoring.

Example

Adjust sentiment weights based on negations (“not good”) or sarcasm cues (“yeah right”).

Q3(b) *Buggy Code: A provided TensorFlow script has errors (e.g., dimension mismatches, incorrect loss functions). Debug and fix the code.*

Troubleshooting challenge:

```python
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, input_shape=(28, 28), activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5)
```

Issues:

- Input shape mismatch- The input should be (28,28,1) or (784) flattened, not (28,28)
- Loss mismatch- Using categorical_crossentropy but y-train likely contains integer labels instead of one-hot vectors.

```python
import tensorflow as tf
from tensorflow.keras import layers, models
# Fix: Flatten input properly for Dense layers
model = models.Sequential([
    layers.Flatten(input_shape=(28, 28)),   # Correct input shape
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
# Fix: Use correct loss for integer labels
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
# Example dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
# Normalize
x_train, x_test = x_train / 255.0, x_test / 255.0
# Train
model.fit(x_train, y_train, epochs=5, validation_split=0.1)
```

Fixed Version

- Flattened input (Flatten() layer)
- Correct loss function (sparse_categorical_crossentropy)
- Normalized data for stable training.