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

TensorFlow:

- Developed by Google.
- Offers static computation graphs (TensorFlow 1.x) and dynamic graphs via tf.function (TensorFlow 2.x).
- Strong production deployment support (e.g., TensorFlow Serving, TensorFlow Lite).
- Rich ecosystem: Keras, TFX, TensorBoard, TF Hub.

PyTorch:

- Developed by Facebook (Meta).
- Uses dynamic computation graphs (eager execution by default).
- Easier for research and experimentation.
- Excellent Python integration, more “Pythonic” than TensorFlow.

When to Choose:

- **Choose TensorFlow**: If your goal is **scalability and deployment** in production (e.g., mobile apps, embedded systems).
- **Choose PyTorch**: If you're doing **research or rapid prototyping**, thanks to its simplicity and flexibility.

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

1. Interactive Data Exploration and Visualization:

- Jupyter Notebooks allow developers to interactively explore datasets by executing code in cells and visualizing results inline.
- Common libraries like matplotlib, seaborn, and plotly integrate seamlessly.
- Ideal for tasks like cleaning data, plotting distributions, and feature engineering.

Example: Plotting the distribution of iris species or visualizing MNIST digit samples before training.

---

2. Model Prototyping and Experimentation:

- Developers can quickly test different machine learning models and tuning parameters in a cell-based workflow.
- Notebooks support real-time feedback, making it easy to compare performance metrics and iterate.

- Great for testing various architectures or hyperparameters before finalizing a script for deployment.

Example: Experimenting with different CNN layers on MNIST to reach 95%+ accuracy.

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

Answer:

1. Linguistic Intelligence:

- spaCy provides advanced NLP features like tokenization, part-of-speech (POS) tagging, named entity recognition (NER), dependency parsing, and lemmatization.
- These features are backed by pre-trained models, enabling accurate analysis of language structure.
- In contrast, basic Python string methods (.split(), .find(), .replace()) treat text as raw sequences without understanding grammar or context.

2. Efficiency and Accuracy:

- spaCy is optimized for speed and performance — it's fast even on large documents.
- Built-in pipelines are trained on large corpora, offering much higher accuracy than custom string rules.
- Also supports adding custom rules or models when needed.

Comparative Analysis: Scikit-learn vs. TensorFlow

Feature	Scikit-learn	TensorFlow
Target Applications	Classical Machine Learning (e.g., SVMs, decision trees, logistic regression)	Deep Learning (e.g., neural networks, CNNs, RNNs)
Ease of Use for Beginners	Very beginner-friendly. Simple APIs with quick results. Great for introductory ML.	More complex, especially for deep learning. TensorFlow 2.x with Keras has improved ease of use.
Community Support	Large and active community, especially for data science and education.	Massive global support from researchers, developers, and enterprises. Extensive tutorials and resources.
Model Complexity	Best for simpler, explainable models.	Suited for complex, high-performance models and large-scale datasets.

Feature	Scikit-learn	TensorFlow
Production Readiness	Limited support for deployment tools.	Strong production ecosystem (TensorFlow Serving, TF Lite, TF.js).
Visualization	Integrates with matplotlib, seaborn for plotting.	Includes TensorBoard for powerful training visualization.

Summary:

- **Use Scikit-learn** for fast prototyping, traditional models, and datasets where interpretability matters.
- **Use TensorFlow** when building and deploying deep learning models at scale or on specialized hardware.

ETHICS & DEBUGGING

Ethical Reflection

1. Bias in MNIST:

- Dataset mainly contains digits in uniform style; may underperform on handwriting from other cultures.

2. Bias in Amazon Reviews:

- Sentiment analysis may misinterpret sarcasm, idioms, or minority dialects.

Mitigation:

- Use TensorFlow’s Fairness Indicators to measure subgroup performance.

- Use spaCy’s rule-based matching to include more linguistic patterns and reduce misclassification bias.

OUTPUT SCREENSHOTS

```
Accuracy: 100.00%
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
```

```
Review 1: I absolutely love the Apple AirPods. The sound quality is amazing!
Named Entities (Product/Brand):
 - the Apple AirPods (PRODUCT)

Review 2: The Samsung Galaxy S21 is overpriced and underperforms. Very disappointed.
Named Entities (Product/Brand):

Review 3: Sony headphones are great for the price. Good bass and noise cancellation.
Named Entities (Product/Brand):
 - Sony (ORG)

Review 4: Terrible experience with the Lenovo ThinkPad. It crashed on day one.
Named Entities (Product/Brand):
 - day one (DATE)

Review 5: I'm impressed with the JBL Flip 5 speaker. It's loud and waterproof!
Named Entities (Product/Brand):
 - JBL Flip (PRODUCT)
 - 5 (CARDINAL)
```

[12]:

```
review_lower = review.lower()

pos_score = sum(word in review_lower for word in positive_words)
neg_score = sum(word in review_lower for word in negative_words)

if pos_score > neg_score:
    sentiment = "Positive"
elif neg_score > pos_score:
    sentiment = "Negative"
else:
    sentiment = "Neutral"

print(f"Sentiment: {sentiment}")
```

```
Sentiment: Positive
```

```
    print(f"Epoch {epoch+1}, Loss: {running_loss / len(trainloader):.4f}")
```

```
Epoch 1, Loss: 0.1245
Epoch 2, Loss: 0.0456
Epoch 3, Loss: 0.0325
Epoch 4, Loss: 0.0249
Epoch 5, Loss: 0.0210
```

[17]:

```
model.eval()
correct = 0
total = 0
with torch.no_grad():
    for images, labels in testloader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

accuracy = correct / total * 100
print(f"\nTest Accuracy: {accuracy:.2f}%")
```

```
Test Accuracy: 98.99%
```

[18]:

```
import numpy as np

def imshow(img):
    img = img * 0.3081 + 0.1307  # Unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)), cmap='gray')
    plt.axis('off')
    plt.show()

dataiter = iter(testloader)
images, labels = next(dataiter)
images, labels = images.to(device), labels.to(device)
```

[19]:

```
outputs = model(images[:5])
_, preds = torch.max(outputs, 1)

print("Predicted:", preds.cpu().numpy())
print("Actual:    ", labels[:5].cpu().numpy())
imshow(torchvision.utils.make_grid(images[:5].cpu()))
```

```
Predicted: [7 2 1 0 4]
Actual:    [7 2 1 0 4]
```