Short Answer Questions

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

- **TensorFlow** uses static computational graphs, which means the graph is defined first and then executed, making it well-suited for deployment in production environments due to its performance optimizations and scalability.
- **PyTorch** employs dynamic computational graphs, allowing for more flexibility and easier debugging during development, which is beneficial for research and experimentation.
- **Choose TensorFlow** when deploying models at scale, especially in production, or if integrating with the TensorFlow ecosystem.
- **Choose PyTorch** for research, rapid prototyping, and when flexibility in model design is a priority.

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

- **Exploratory Data Analysis (EDA):** Jupyter notebooks allow data scientists to visualize data, run code interactively, and document their analysis process step-by-step.
- **Model Development and Experimentation:** They enable quick testing of machine learning models, hyperparameter tuning, and visualization of training progress in an integrated environment.

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

- **spaCy** provides pre-built models and pipelines for common NLP tasks like tokenization, part-of-speech tagging, named entity recognition, and dependency parsing, greatly simplifying and speeding up development.
- **Basic Python string operations** require manual implementation of NLP tasks, which can be error-prone and less efficient for complex language processing.

Comparative Analysis

Aspect	**Scikit-learn**	**TensorFlow**
Target Applications	Classical machine learning (e.g., regression, classification, clustering)	Deep learning (e.g., neural networks, CNNs, RNNs)
Ease of Use for Beginners	Easier to learn with simple API	Steeper curve, but well-documented

Aspect	Scikit-learn	TensorFlow
Community Support	Large, mature community, extensive tutorials	Very large, active community, extensive resources

When analyzing the MNIST model, potential biases can manifest due to imbalanced digit representation, or correlations between digit and background features (e.g., color or position biases in datasets like Colored MNIST). Research shows bias in deep convolutional neural networks can sometimes be detected automatically by examining the model's weights without running inference, achieving high accuracy in bias detection. This suggests that bias is encoded in model parameters themselves, which opens new ways to identify and mitigate bias through weight analysis.

Using tools like TensorFlow Fairness Indicators, one can measure and visualize disparities in model performance across different digit classes or demographic slices if available, ensuring the model does not favor certain digits or features disproportionately. SpaCy's rule-based systems can assist in detecting biased or imbalanced language patterns in text-based data such as Amazon Reviews, helping to preemptively correct skewed or unfair data inputs before training.

Debugging TensorFlow code for these models often involves resolving issues like dimension mismatches between layers or incorrect loss functions, which can be fixed by verifying tensor shapes and using appropriate loss formulations (e.g., sparse categorical cross-entropy for classification tasks).

In essence, these approaches together—weight inspection for bias detection, fairness metrics evaluation, rule-based language filtering, plus careful debugging—form a comprehensive strategy for ethically responsible and technically robust AI models.