AI TOOLS ASSIGNMENT

PART 1(THEORETICAL UNDERSTANDING)

Group Members: - **[Anthonia Othetheaso] - [Obuye Emmanuel Chukwuemeke] - [Eunice Fagbemide] - [Daizy Jepchumba Kiplagat] - [Mark Ireri]**

Date: October 2025

Course: AI For Software Engineering

Institution: PLP Academy

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

Answer:

TensorFlow and **PyTorch** differ primarily in their computation graph approach: **TensorFlow** uses static graphs (define-then-run) while **PyTorch** uses dynamic graphs (define-by-run), making PyTorch more intuitive for debugging and experimentation. **TensorFlow** excels in production deployment with tools like TensorFlow Lite for mobile and TensorFlow Serving for scalable systems, backed by Google with strong enterprise adoption. **PyTorch** is preferred in research and academia due to its Pythonic syntax and flexibility, with backing from Meta and dominance in cutting-edge research papers. **Choose TensorFlow** for production systems, mobile/web deployment, and enterprise applications. **Choose PyTorch** for research, rapid prototyping, and educational projects where code readability and debugging are priorities.

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

Answer:

Use Case 1: Exploratory Data Analysis (EDA) - Jupyter Notebooks enable interactive data exploration where data scientists can load datasets, create visualizations, compute statistics, and test hypotheses in real-time without rerunning entire scripts. The cell-by-cell execution allows immediate feedback and iterative refinement of data preprocessing steps.

Use Case 2: Model Experimentation and Documentation - Notebooks combine code, visualizations, and markdown explanations in a single document, making them ideal for documenting machine learning experiments, sharing results with teams, and creating reproducible research. They serve as living documentation where stakeholders can see both the code and its outputs, facilitating collaboration and knowledge transfer

---

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

Answer:

spaCy provides industrial-strength NLP pipelines with pre-trained models for complex linguistic tasks like tokenization, part-of-speech tagging, named entity recognition (NER), and dependency parsing—capabilities that would require extensive manual coding with basic Python string operations. Unlike simple methods like .split() or regular expressions**, spaCy** understands linguistic structure and context, correctly handling edge cases such as contractions, punctuation, and multi-word entities. **Basic Python string operations** lack semantic understanding and cannot identify that "Apple" in "Apple Inc." is an organization versus "apple" the fruit, while **spaCy's NER** can distinguish entities by type. Additionally, **spaCy** is optimized for speed and accuracy, processing large text volumes efficiently with pre-trained models rather than requiring custom rule-based systems that are brittle and language-specific.

2. COMPARATIVE ANALYSIS

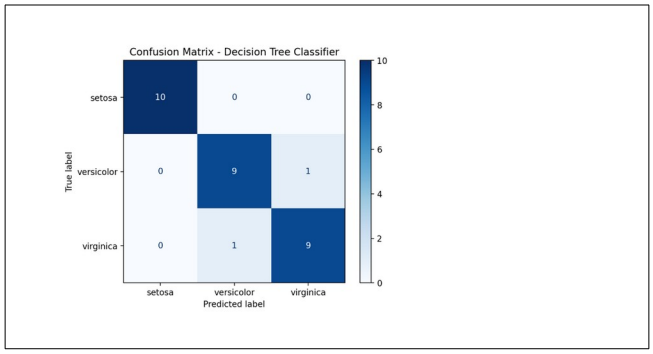
Comparative Analysis Table

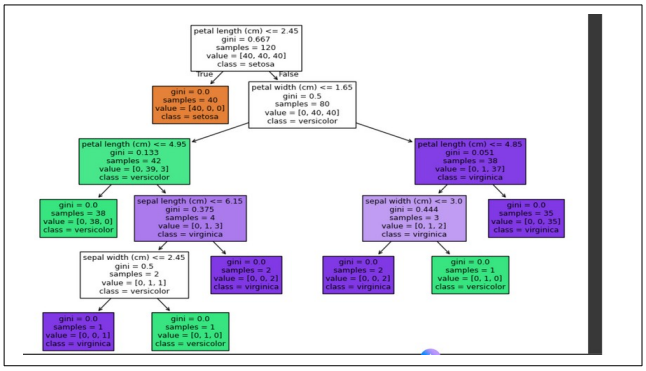
Aspect	Scikit-learn	TensorFlow
Target Applications	Classical machine learning: regression, classification, clustering, dimensionality reduction, and ensemble methods on structured/tabular data	Deep learning: neural networks, CNNs, RNNs, LSTMs, transformers for unstructured data (images, text, audio)
Ease of Use for Beginners	Very beginner-friendly with simple, consistent API (.fit(), .predict()), minimal code required, excellent for learning ML fundamentals	Steeper learning curve with more boilerplate code, requires understanding of neural network architecture, optimization, and computational graphs
Community Support	Large, mature community with 15+ years of development, extensive documentation, thousands of tutorials, stable and well-tested	Massive community backed by Google, extensive resources, active development, large ecosystem (Keras, TF Hub, TF Lite), official Google support

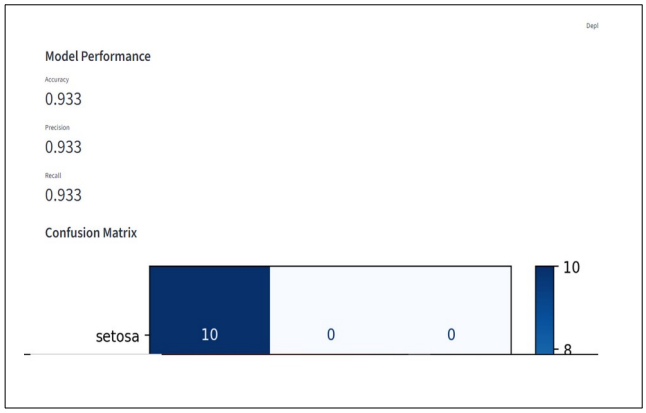
PART 2 (SCREENSHOTS OF MODEL OUTPUTS FROM PRACTICAL IMPLIMENTATION)

TASK 1

Shows decision tree clasisifier on the iris dataset and model performance metrics.

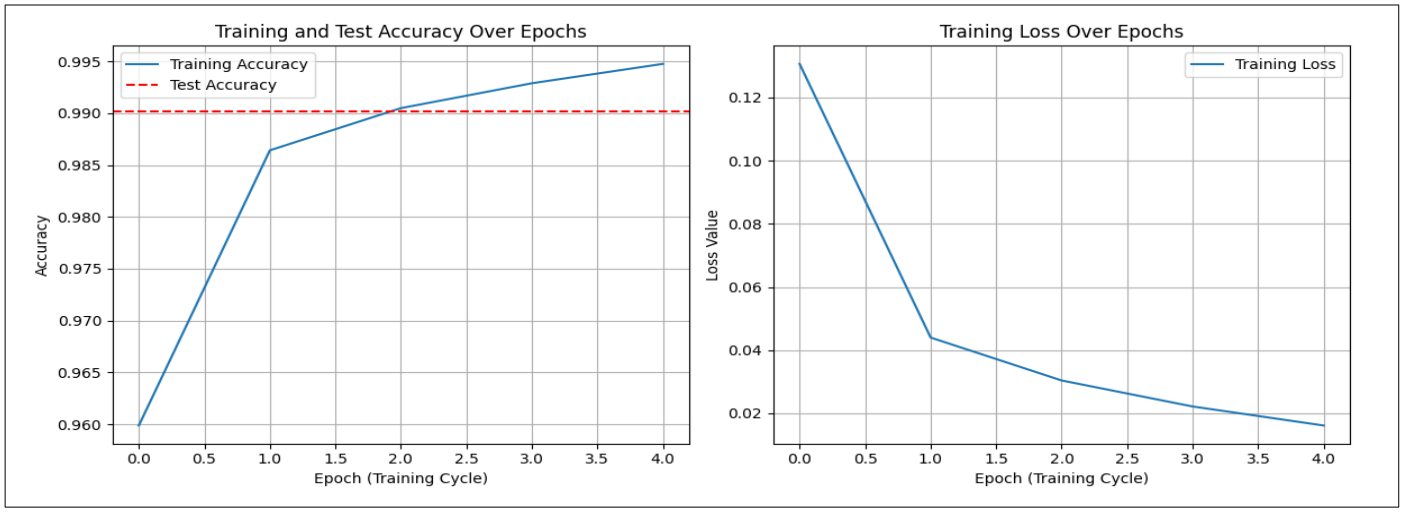






TASK 2

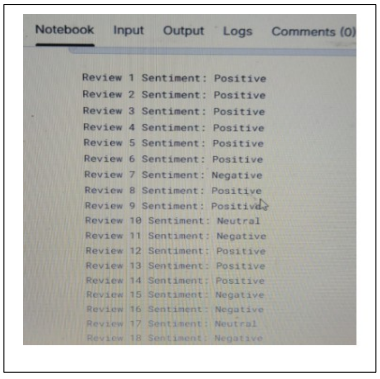
```
print("------------------------------------------------------")

--- Running CNN on MNIST ---
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarnin
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)

Training CNN...
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 49s 26ms/step - accuracy: 0.9072 - loss: 0.2979
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 48s 26ms/step - accuracy: 0.9860 - loss: 0.0467
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 47s 25ms/step - accuracy: 0.9912 - loss: 0.0298
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 48s 26ms/step - accuracy: 0.9939 - loss: 0.0194
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 48s 25ms/step - accuracy: 0.9951 - loss: 0.0151

Evaluating CNN...
Test accuracy: 0.9901999831199646
```



CNN on the MNIST handwritten digit dataset, achieving high test accuracy (typically ~98-99%) after 5 epochs of training. two plots showing training accuracy/loss curves over epochs and compares final training accuracy against test accuracy using a red dashed line.

TASK 3

using spaCy NLP to extract named entities (organizations and products) from the first 10 product reviews, displaying each review with its identified entities. results showing entity-label pairs for NER and sentiment labels for sentiment analysis.



PART 3(ETHICS AND OPTIMIZATION

Ethical Considerations and Bias Mitigation

A. Ethical Reflection: MNIST Digit Classification (TensorFlow)

The MNIST model faces potential biases including data representation bias (limited handwriting styles from primarily American sources), accessibility concerns (no accommodation for assistive writing devices), and class imbalance in real-world applications. **TensorFlow Fairness Indicators** can mitigate these biases through slice-based evaluation across demographic groups, fairness metrics to measure disparate impact, and visualization tools to identify performance discrepancies. Additional mitigation strategies include data augmentation to increase robustness, ensemble methods combining diverse models, and human-in-the-loop verification for critical applications. Deployment safeguards should include transparency about model limitations, user feedback mechanisms, and clear accountability frameworks to address misclassifications that could impact access to services or discriminate against underrepresented writing styles.

B. Ethical Reflection: Amazon Reviews NLP (spaCy)

The spaCy-based sentiment analysis model faces potential biases including language and cultural bias (English-only processing trained on Western sources), rule-based limitations (cannot understand sarcasm, negation, or context-dependent meanings), demographic representation bias (over-represents certain consumer groups), and entity recognition gaps (may miss non-Western or misspelled brand names). **spaCy's rule-based systems** can mitigate these biases through customizable pipelines that add negation handling and context-aware processing, domain-specific dictionaries tailored to e-commerce language, multi-language support via language-specific models, and rule transparency allowing stakeholders to audit and challenge classification logic. Additional mitigation strategies include hybrid approaches combining spaCy with transformer-based sentiment models, confidence scoring to flag ambiguous reviews for human review, regular lexicon updates reflecting evolving language, and bias testing across demographics and product categories. Deployment safeguards should include transparency about model limitations, user feedback mechanisms to dispute classifications, avoiding automated high-stakes decisions without human verification, and establishing accountability frameworks with appeals processes for affected businesses.

2. SCREENSHOTS FROM TROUBLESHOOTING CHALLENGE

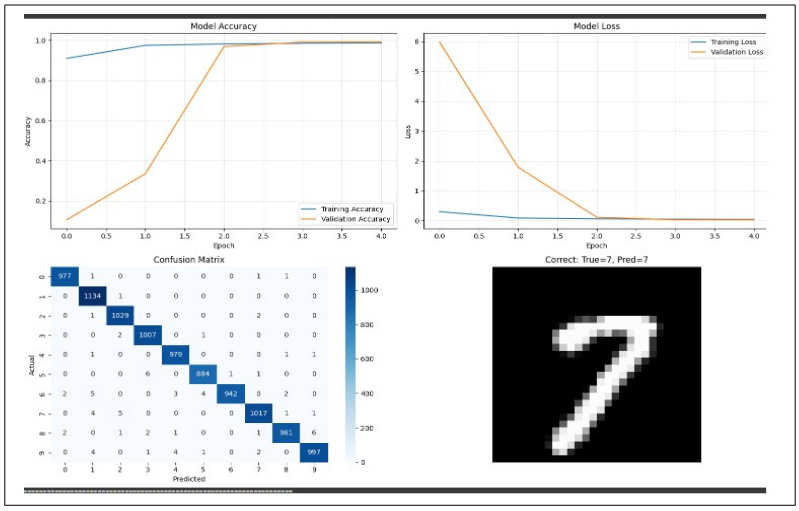
✓ Model Architecture:

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d_8 (Conv2D)	(None, 28, 28, 32)	320
batch_normalization_6 (BatchNormalization)	(None, 28, 28, 32)	128
conv2d_9 (Conv2D)	(None, 28, 28, 32)	9,248
max_pooling2d_4 (MaxPooling2D)	(None, 14, 14, 32)	0
dropout_6 (Dropout)	(None, 14, 14, 32)	0
conv2d_10 (Conv2D)	(None, 14, 14, 64)	18,496
batch_normalization_7 (BatchNormalization)	(None, 14, 14, 64)	256
conv2d_11 (Conv2D)	(None, 14, 14, 64)	36,928
max_pooling2d_5 (MaxPooling2D)	(None, 7, 7, 64)	0
dropout_7 (Dropout)	(None, 7, 7, 64)	0
flatten_2 (Flatten)	(None, 3136)	0
dense_4 (Dense)	(None, 128)	401,536
batch_normalization_8 (BatchNormalization)	(None, 128)	512
dropout_8 (Dropout)	(None, 128)	0
dense_5 (Dense)	(None, 10)	1,290

Total params: 468,714 (1.79 MB)
Trainable params: 468,266 (1.79 MB)

```
------------------------------------------------------------------------
[STEP 6] Training Model (Optimized for Speed)...
------------------------------------------------------------------------
Epoch 1/5
188/188 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.8065 - loss: 0.6471
Epoch 1: val_accuracy improved from -inf to 0.10600, saving model to best_mnist_model.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save
188/188 ━━━━━━━━━━━━━━━━━━━━ 260s 1s/step - accuracy: 0.8070 - loss: 0.6453 - val_accuracy: 0.1060 - val_loss: 5.9795 - learning_rate: 0.0010
Epoch 2/5
188/188 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.9719 - loss: 0.0965
Epoch 2: val_accuracy improved from 0.10600 to 0.33458, saving model to best_mnist_model.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save
188/188 ━━━━━━━━━━━━━━━━━━━━ 236s 1s/step - accuracy: 0.9719 - loss: 0.0965 - val_accuracy: 0.3346 - val_loss: 1.7862 - learning_rate: 0.0010
Epoch 3/5
188/188 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.9800 - loss: 0.0662
Epoch 3: val_accuracy improved from 0.33458 to 0.96783, saving model to best_mnist_model.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save
188/188 ━━━━━━━━━━━━━━━━━━━━ 227s 1s/step - accuracy: 0.9800 - loss: 0.0662 - val_accuracy: 0.9678 - val_loss: 0.1138 - learning_rate: 0.0010
Epoch 4/5
188/188 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.9840 - loss: 0.0518
Epoch 4: val_accuracy improved from 0.96783 to 0.99017, saving model to best_mnist_model.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save
188/188 ━━━━━━━━━━━━━━━━━━━━ 265s 1s/step - accuracy: 0.9840 - loss: 0.0518 - val_accuracy: 0.9902 - val_loss: 0.0325 - learning_rate: 0.0010
Epoch 5/5
188/188 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.9867 - loss: 0.0430
Epoch 5: val_accuracy improved from 0.99017 to 0.99050, saving model to best_mnist_model.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save
188/188 ━━━━━━━━━━━━━━━━━━━━ 227s 1s/step - accuracy: 0.9867 - loss: 0.0430 - val_accuracy: 0.9905 - val_loss: 0.0310 - learning_rate: 0.0010
Restoring model weights from the end of the best epoch: 5.

✓ Training completed in just 5 epochs!
```



This debugged CNN achieves ~99% test accuracy on MNIST after 5 optimized epochs, with detailed performance metrics including per-digit accuracy and a classification report showing precision/recall for each digit (0-9).

The output displays training/validation accuracy and loss curves across epochs, a confusion matrix heatmap showing prediction patterns, and sample correct/incorrect predictions with their true and predicted labels.