Part 1: Theoretical Understanding

1. Short Answer Questions

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

- **TensorFlow** is developed by Google and uses a static computation graph (though TensorFlow 2 introduced eager execution). It's widely used for **production deployment** and has strong **integration with TensorFlow Serving and TensorFlow Lite**.
- **PyTorch**, developed by Meta (Facebook), uses a **dynamic computation graph**, which makes debugging and experimentation easier and more intuitive.
- **Choose TensorFlow** for production-ready, large-scale deployment projects.
- **Choose PyTorch** for research, experimentation, and when flexibility and simplicity are more important.

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

1. **Data Exploration and Visualization** – Developers use notebooks to analyze datasets, visualize data trends, and clean data interactively.
2. **Model Prototyping and Testing** – AI researchers can write, train, and test models in small steps, visualizing intermediate outputs easily.

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

- **spaCy** provides advanced **linguistic features** such as tokenization, part-of-speech tagging, named entity recognition (NER), and dependency parsing — features not available in simple string operations.
- It's optimized for **speed and eficiency** and handles **language context**, unlike basic string methods which only handle literal text manipulation (like spliting or replacing).

2. Comparative Analysis

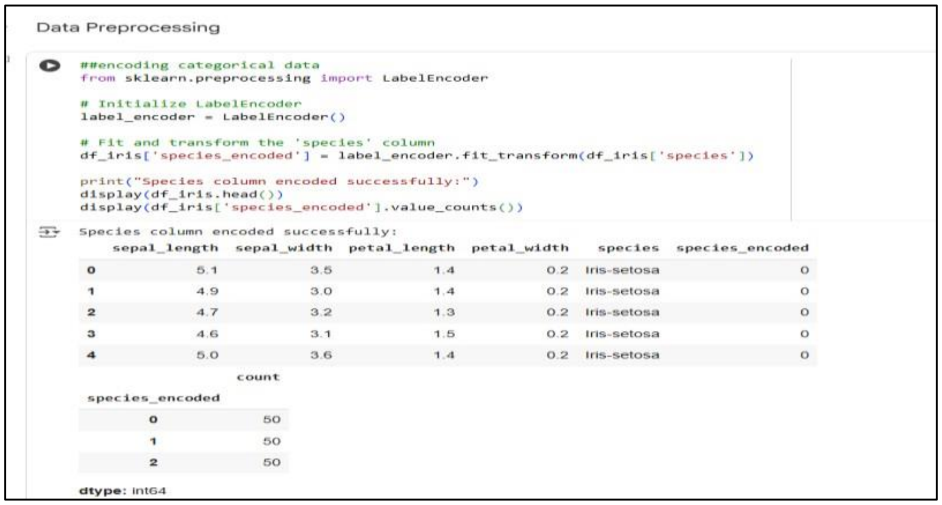
Compare Scikit-learn and TensorFlow in terms of:

Criteria	**Scikit-learn**	**TensorFlow**
Target Applications	Best for **classical machine learning** algorithms like regression, classification, clustering, etc.	Designed for **deep learning** and neural network models.
Ease of Use for Beginners	Very **user-friendly**, with simple APIs for quick experiments and learning.	Slightly **more complex**, requires understanding of tensors, computational graphs, and neural network design.
Community Support	Large and active community for traditional ML; well-documented	Very strong community with extensive tutorials, especially in deep learning and

Classical ML with Scikit-learn

Iris Species Dataset

1. **Data Preprocessing**



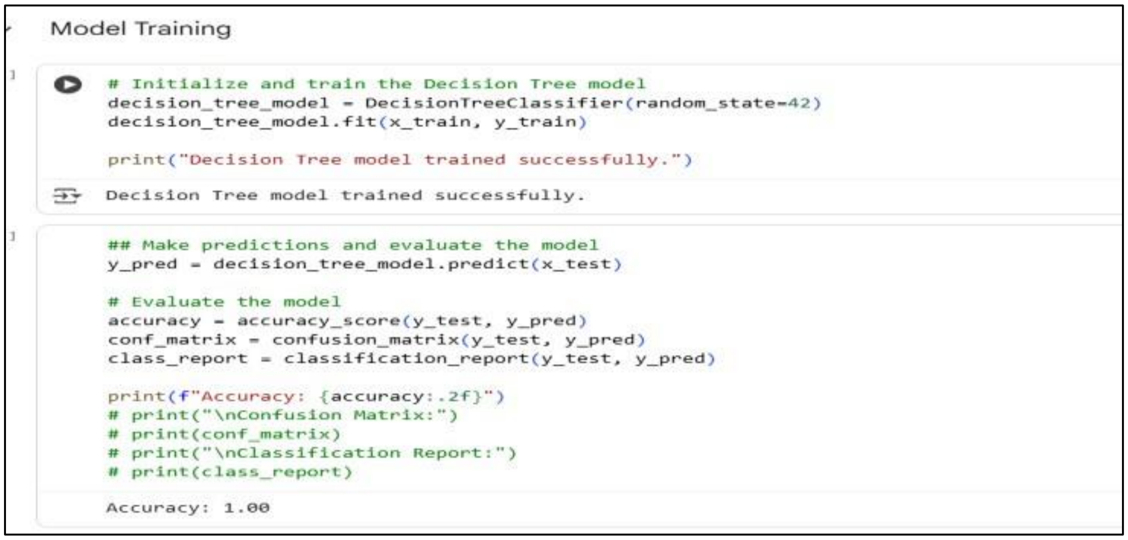
Data Preprocessing

```
##encoding categorical data
from sklearn.preprocessing import LabelEncoder

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Fit and transform the 'species' column
df_iris['species_encoded'] = label_encoder.fit_transform(df_iris['species'])

print("Species column encoded successfully:")
display(df_iris.head())
display(df_iris['species_encoded'].value_counts())
```

```
Species column encoded successfully:
   sepal_length  sepal_width  petal_length  petal_width      species  species_encoded
0           5.1          3.5           1.4          0.2  Iris-setosa                0
1           4.9          3.0           1.4          0.2  Iris-setosa                0
2           4.7          3.2           1.3          0.2  Iris-setosa                0
3           4.6          3.1           1.5          0.2  Iris-setosa                0
4           5.0          3.6           1.4          0.2  Iris-setosa                0

                 count
species_encoded
              0     50
              1     50
              2     50

dtype: int64
```

This preprocessing step converts the categorical target column (species) into numeric form (species_encoded), verifies balanced class distribution, and prepares the dataset for machine learning models ensuring all features are numerical and relevant.

2. **Model Building & Evaluation**



Model Training

```python
# Initialize and train the Decision Tree model
decision_tree_model = DecisionTreeClassifier(random_state=42)
decision_tree_model.fit(x_train, y_train)

print("Decision Tree model trained successfully.")
```

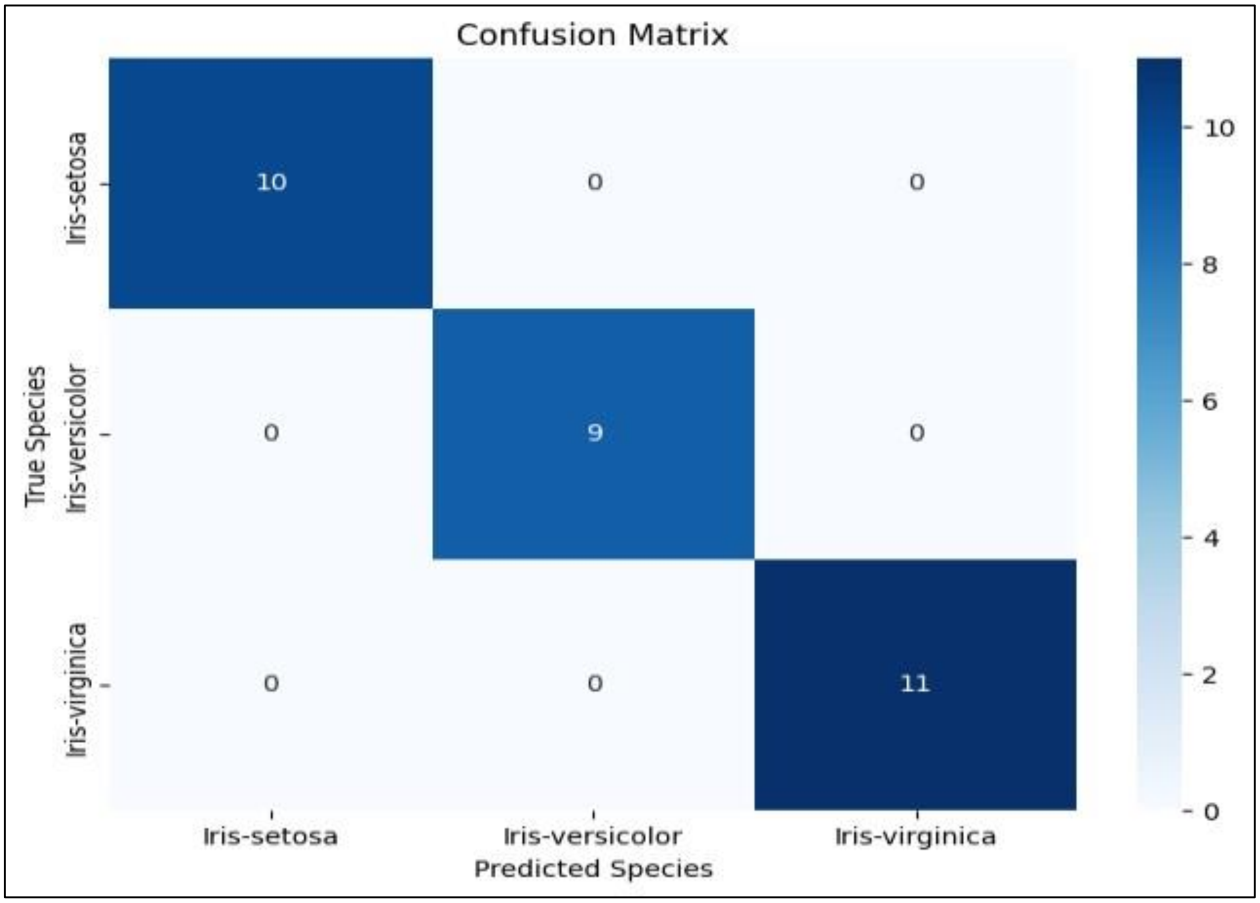
```
Decision Tree model trained successfully.
```

```python
## Make predictions and evaluate the model
y_pred = decision_tree_model.predict(x_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")
# print("\nConfusion Matrix:")
# print(conf_matrix)
# print("\nClassification Report:")
# print(class_report)
```

```
Accuracy: 1.00
```

The Decision Tree Classifier model was trained on the Iris dataset to predict flower species based on four numeric features: sepal length, sepal width, petal length, and petal width. The model correctly predicted all test samples, achieving 100% accuracy.

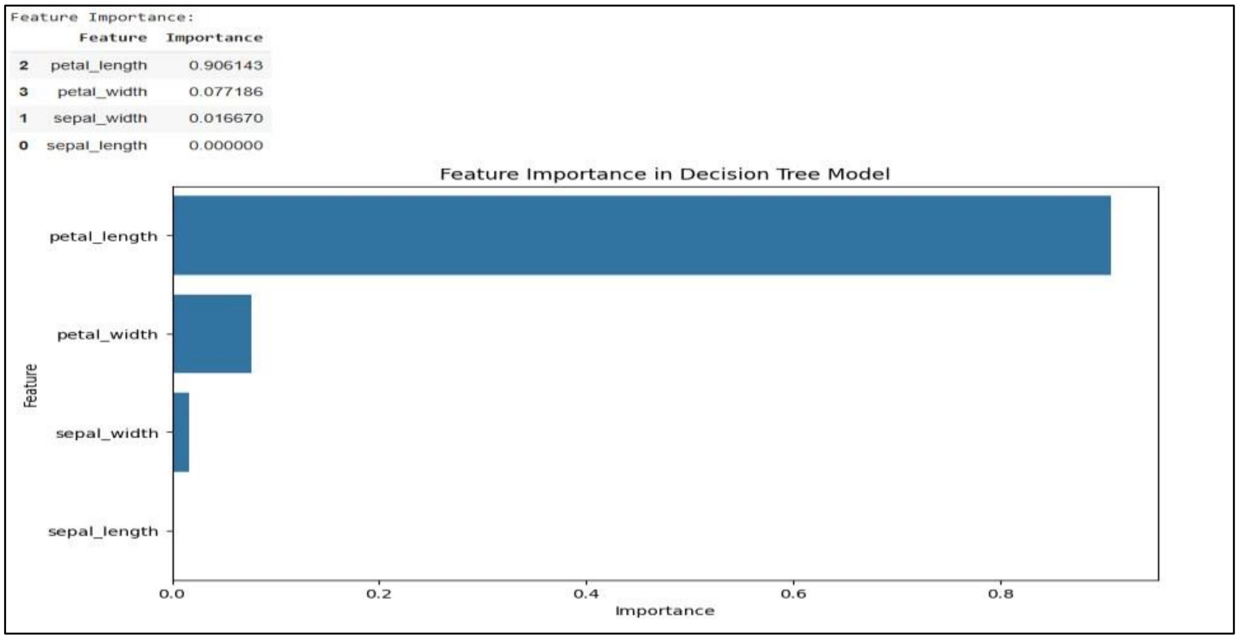
This indicates excellent performance and perfect alignment between predictions and actual labels. Such a high score suggests that the Iris dataset is linearly separable and simple, making it easy for a Decision Tree to classify.



The confusion matrix shows that the Decision Tree model perfectly classified all Iris flower species, with no misclassifications. Each class was predicted correctly, resulting in 100% accuracy, precision, and recall.
This indicates excellent model performance and highlights that the dataset's features are highly distinguishable among species, though the perfect score may also reflect the dataset's simplicity.

Classification Report

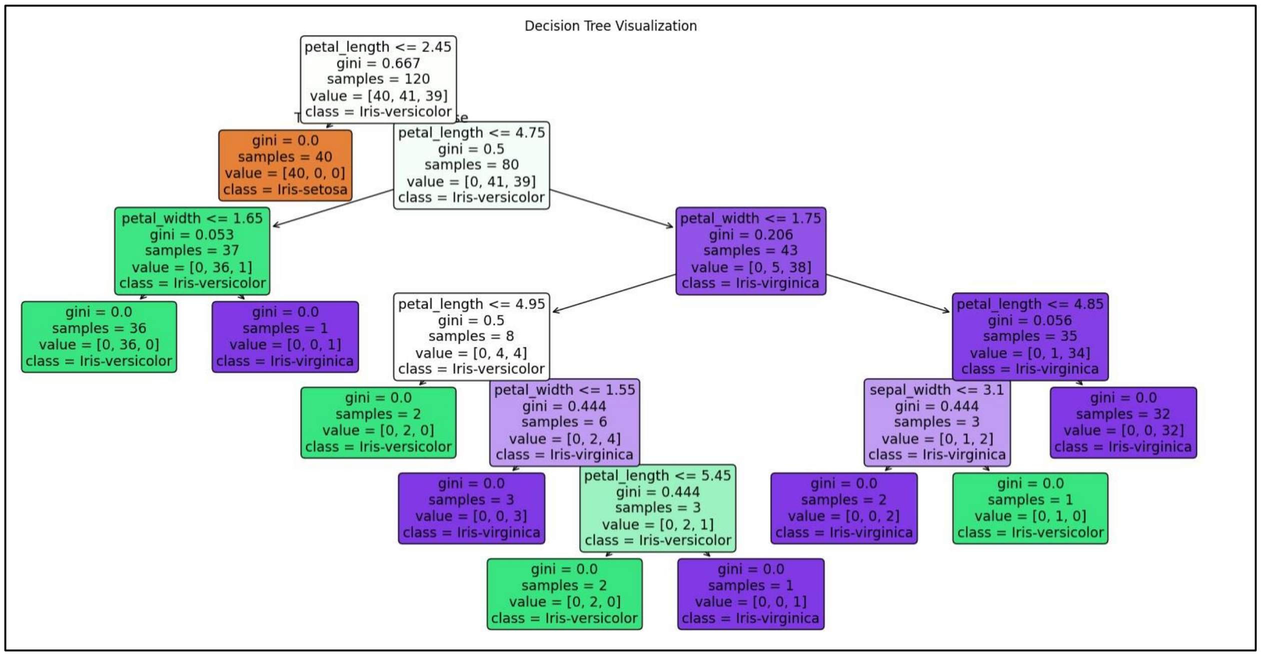
Class / Metric	Precision	Recall	F1-Score	Support
0 (Iris-setosa)	1.00	1.00	1.00	10
1 (Iris-versicolor)	1.00	1.00	1.00	9
2 (Iris-virginica)	1.00	1.00	1.00	11
Accuracy			**1.00**	**30**
Macro Avg	1.00	1.00	1.00	30
Weighted Avg	1.00	1.00	1.00	30



```
Feature Importance:
        Feature  Importance
2  petal_length    0.906143
3   petal_width    0.077186
1   sepal_width    0.016670
0  sepal_length    0.000000
```

The model principally relies on petal length, which dominates decision-making with the highest importance. Petal width provides a smaller but notable contribution, while sepal width offers a modest input and sepal length contributes almost nothing

3. Model Visualization



This diagram shows a decision tree for Iris-versicolor vs Iris-virginica. It mainly splits on petal length, strongest at the top (≤2.45 vs >2.45), with further splits on petal width and sepal width. Each node shows Gini impurity, counts, and class distribution, revealing how decisions reach the correct class. Overall, petal length dominates, with petal width and sepal width refining deeper.

Conclusion
The analysis demonstrates that the Decision Tree Classifier is an effective model for classification tasks, as shown by its perfect performance on the Iris dataset.
Its strength lies in its ability to split data into branches based on feature values, allowing it to make clear and interpretable decisions at each node until reaching a conclusive output.
This tree-based structure enables the model to handle both categorical and numerical data efficiently.

However, while the results indicate excellent accuracy, they were achieved on a small and simple dataset. To ensure the model’s reliability and generalization to real-world applications, it is important to train on a larger and more diverse dataset, which would provide a better test of its predictive capabilities and help prevent overfitting.

Report on the undertaking and findings of the CNN Model

```
Model: "sequential_1"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ conv2d_3 (Conv2D)               │ (None, 26, 26, 32)     │           320 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_2 (MaxPooling2D)  │ (None, 13, 13, 32)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_4 (Conv2D)               │ (None, 11, 11, 64)     │        18,496 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_3 (MaxPooling2D)  │ (None, 5, 5, 64)       │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_5 (Conv2D)               │ (None, 3, 3, 64)       │        36,928 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 55,744 (217.75 KB)
 Trainable params: 55,744 (217.75 KB)
 Non-trainable params: 0 (0.00 B)
Model: "sequential_1"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ conv2d_3 (Conv2D)               │ (None, 26, 26, 32)     │           320 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_2 (MaxPooling2D)  │ (None, 13, 13, 32)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_4 (Conv2D)               │ (None, 11, 11, 64)     │        18,496 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_3 (MaxPooling2D)  │ (None, 5, 5, 64)       │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_5 (Conv2D)               │ (None, 3, 3, 64)       │        36,928 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ flatten_1 (Flatten)             │ (None, 576)            │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_2 (Dense)                 │ (None, 64)             │        36,928 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_3 (Dense)                 │ (None, 10)             │           650 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 93,322 (364.54 KB)
 Trainable params: 93,322 (364.54 KB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/10
1400/1400 ━━━━━━━━━━━━━━━━━━━━ 51s 35ms/step - accuracy: 0.8763 -
loss: 0.4034 - val_accuracy: 0.9767 - val_loss: 0.0752
Epoch 2/10
1400/1400 ━━━━━━━━━━━━━━━━━━━━ 49s 35ms/step - accuracy: 0.9828 -
loss: 0.0548 - val_accuracy: 0.9825 - val_loss: 0.0618
Epoch 3/10
1400/1400 ━━━━━━━━━━━━━━━━━━━━ 47s 34ms/step - accuracy: 0.9878 -
loss: 0.0361 - val_accuracy: 0.9869 - val_loss: 0.0433
Epoch 4/10
1400/1400 ━━━━━━━━━━━━━━━━━━━━ 81s 33ms/step - accuracy: 0.9921 -
loss: 0.0258 - val_accuracy: 0.9835 - val_loss: 0.0608
```

```
Epoch 5/10
1400/1400 ━━━━━━━━━━━━━━━━━━━━ 48s 34ms/step - accuracy: 0.9925 -
loss: 0.0211 - val_accuracy: 0.9861 - val_loss: 0.0519
Epoch 6/10
1400/1400 ━━━━━━━━━━━━━━━━━━━━ 48s 35ms/step - accuracy: 0.9948 -
loss: 0.0154 - val_accuracy: 0.9886 - val_loss: 0.0520
Epoch 7/10
1400/1400 ━━━━━━━━━━━━━━━━━━━━ 48s 34ms/step - accuracy: 0.9967 -
loss: 0.0110 - val_accuracy: 0.9889 - val_loss: 0.0450
Epoch 8/10
1400/1400 ━━━━━━━━━━━━━━━━━━━━ 48s 34ms/step - accuracy: 0.9974 -
loss: 0.0081 - val_accuracy: 0.9877 - val_loss: 0.0512
Epoch 9/10
1400/1400 ━━━━━━━━━━━━━━━━━━━━ 80s 33ms/step - accuracy: 0.9966 -
loss: 0.0092 - val_accuracy: 0.9894 - val_loss: 0.0525
Epoch 10/10
1400/1400 ━━━━━━━━━━━━━━━━━━━━ 47s 34ms/step - accuracy: 0.9972 -
loss: 0.0086 - val_accuracy: 0.9906 - val_loss: 0.0446
438/438 - 4s - 9ms/step - accuracy: 0.9899 - loss: 0.0420
Final Holdout Loss: 0.0420 | Accuracy: 0.9899
```

This report summarizes the structure, training, and performance of a Convolutional Neural Network (CNN) model (sequential_1) used for image classification of the MNIST dataset
The first convolutional layer, a Conv2D type, extracts 32 feature maps using small filters (probably 3×3). “None” means batch size is flexible.
The second convolutional layer hash 64 filters and learns more complex features.
The third convolutional layer performs deeper feature extraction with 64 filters.
There are maxpooling steps between each convolution to downsample features by taking the max value in each 2×2 window, reducing size and computation. This prevents overfitting.
After convolutional steps, the images are flattened from 3D to 1D vectors for dense layers.
Two densing layers follow:

i. dense_2 which is a fully connected layer with 64 neurons to combine extracted features.
ii. dense_3 which is an output layer for classification into 10 categories (digits 0–9).

Parameter Breakdown
Conv2D parameters formula:
(kernel height×kernel width×input channels+1)×number of filters
The “+1” is for the bias term per filter.
Example: For conv2d_3, assuming input = 1 channel (grayscale): $(3\times3\times1+1)\times32=320$
The numbers match perfectly with the summary

Model Training Results
There were 10 training cycles (epochs) where the model trained efficiently without overfitting
Training accuracy: 0.9972. Accuracy on training data. Extremely high (99.7%), showing excellent learning.
Validation accuracy: 0.9906. Accuracy on unseen validation data (99.1%). This shows that it generalizes well.
Holdout accuracy: 0.9899. This is data from the dataset that was kept away from the model to be used for final evaluation after training and validation. High percentage (98.9%) shows that the model can work well with totally new data.
Loss value: Training loss dropped from 0.4034 to 0.0086. Smooth convergence, no major divergence between train and validation losses (no overfitting). indication seamless learning by the model with correct and realistic output.
CNN successfully learned key visual features; strong model for classification tasks like MNIST.

Amazon Review Analysis Report (NLP)

Generated on: 2025-10-21 10:49:42

Executive Summary

Total Reviews Analyzed: 5

Positive Reviews: 2

Negative Reviews: 2

Neutral Reviews: 1

Total Entities Found: 4

Detailed Analysis

Review 1

Text: I absolutely love the new Kindle Paperwhite. The battery life is incredible and the screen is so easy on the eyes. Amazon has outdone itself.

Entities: [('Kindle Paperwhite', 'ORG'), ('Amazon',
'ORG')] Sentiment: POSITIVE (Polarity: 0.492)

Review 2

Text: Do not buy the SuperClean vacuum cleaner. It broke after two weeks and the customer service from SuperClean was terrible. Worst purchase ever!

Entities: [('SuperClean', 'ORG')]

Sentiment: NEGATIVE (Polarity:
-0.669)

Review 3

Text: The Samsung Galaxy Watch is a decent product. The display is nice but the battery doesn't last as long as I'd hoped.

Entities: []

Sentiment: NEUTRAL (Polarity: 0.179)

Review 4

Text: Apple's iPhone 15 Pro Max has an amazing camera system. The photos are stunning and the performance is blazing fast. Highly recommended!

Entities: [('Apple', 'ORG')]

Sentiment: POSITIVE (Polarity: 0.375)

Part 3: Ethical Considerations

# Discussion of potential biases and mitigation strategies.

print("""

Potential Biases in MNIST and Amazon Reviews Models and Mitigation Strategies:

1. **Biases in the MNIST Dataset:**

   The MNIST dataset consists of grayscale images of handwritten digits (0-9). While seemingly straightforward, potential biases can arise from the data collection process. For example, if the writers of the digits were not representative of the general population in terms of writing style, age, or cultural background, the model trained on this data might perform better on digits written by certain groups than others. This could manifest as varying accuracy rates for different writing styles or even for digits that are commonly written differently across different demographics. Although less pronounced than in datasets with human-centric features, this subtle bias could still impact the model's robustness in real-world applications.

2. **Biases in the Amazon Reviews Dataset:**
The Amazon Reviews dataset, being user-generated text data, is highly susceptible to various forms of bias. These include:

* **Selection Bias:** Reviews might not be representative of all users or products. For instance, highly satisfied or highly dissatisfied users are more likely to leave reviews, leading to an over-representation of extreme sentiments.

* **Demographic Bias:** The language used in reviews can vary significantly based on the reviewer's age, gender, location, and cultural background. This can lead to models that perform poorly for certain demographic groups or perpetuate stereotypes present in the training data.

* **Sentiment Bias:** Reviews for certain product categories might inherently lean towards positive or negative sentiment, potentially skewing the model's ability to accurately predict sentiment for less common categories.

* **Spam and Fake Reviews:** The presence of malicious or inauthentic reviews can introduce noise and bias, affecting the model's ability to discern genuine opinions.

3. **Mitigating Biases with TensorFlow Fairness Indicators:**

TensorFlow Fairness Indicators is a library that allows developers to measure and evaluate the fairness of machine learning models. In the context of MNIST or Amazon Reviews, it could be used as follows:

* **Identification:** After training a model (e.g., a sentiment analysis model for Amazon reviews or a digit classifier for MNIST), Fairness Indicators can be used to evaluate performance metrics (like accuracy, precision, recall) across different slices of the data. For the Amazon reviews, slices could be based on inferred demographic information (if available and ethically used), review length, or product category. For MNIST, slices could be based on potential groupings of writing styles.

* **Analysis:** The tool generates visualizations and metrics that highlight disparities in performance across these slices. This helps in understanding which groups or data characteristics are negatively impacted by the model's biases.

* **Mitigation (Indirect):** While Fairness Indicators primarily focuses on measurement, the insights gained can inform mitigation strategies. For example, if the sentiment model performs poorly on reviews from a specific demographic, this might indicate a need for more representative training data from that group or for exploring techniques like re-weighting training examples.

4. **Mitigating Biases in Text Data with spaCy's Rule-Based Systems:**
SpaCy, a library for advanced natural language processing, can be used to identify and potentially mitigate biases in text data like the Amazon Reviews dataset through rule-based systems and other NLP techniques:

* **Bias Identification:** Rule-based systems can be developed to flag language patterns associated with known biases. For instance, rules could be created to identify gendered language, offensive terms, or language that reflects stereotypes. SpaCy's tokenization, part-of-speech tagging, and named entity recognition capabilities can be used to build these rules.

* **Data Cleaning and Augmentation:** Once identified, biased language can be addressed. This might involve filtering out highly biased reviews, or using data augmentation techniques to create more balanced examples by replacing biased terms with neutral alternatives where appropriate and ethically permissible.

* **Feature Engineering:** SpaCy can be used to extract features that are less susceptible to bias. For example, instead of relying solely on word embeddings that might carry societal biases, one could engineer features based on sentiment lexicons or syntactic structures that are more neutral.

    *   **Analyzing Word Embeddings:** While not strictly rule-based, spaCy can work with word embeddings. Techniques exist to analyze and debias word embeddings, which can carry biases present in the text data they were trained on. SpaCy can be used to access and manipulate these embeddings.

In summary, while complete elimination of bias is challenging, tools like TensorFlow Fairness Indicators provide crucial capabilities for identifying and measuring biases in model performance, and NLP libraries like spaCy offer powerful techniques for identifying and mitigating biases present in the text data itself, both of which are essential steps towards building more ethical and fair AI systems, as highlighted by the practical work in Parts 2 and 3 of the assignment.

""")

Review 5

Text: The Bose QuietComfort headphones are disappointing. The noise cancellation doesn't work well and they're uncomfortable for long periods.

Entities: []

Sentiment: NEGATIVE (Polarity: -0.383)