WEEK 3
PART 0NE: A REPORT

GROUP 93

GROUP MEMBERS

1. GATHIGI MOSES MUIRURI – gathimoses@gmail.com
2. ODONGO ISAIAH – odongoreagan19@gmail.com
3. KEREN HAPUCH NTINYARI – kerenhapuch68@gmail.com
4. JEBICHII JOYCE – jebichiijoyce@gmail.com
5. PALPABLE SMART – palpable237@gmail.com

WEEK 3
PART ONE: A REPORT

📄 AI Tools and Applications Report

1. Short Answer Questions

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

Answer:
TensorFlow and PyTorch are leading deep learning frameworks with distinct design philosophies and use cases.

Feature	TensorFlow	PyTorch
Execution Model	Static graph (eager execution optional)	Dynamic graph (eager execution default)
Syntax & Usability	More verbose, structured API	Pythonic, intuitive, and flexible
Deployment	Robust (TF Serving, TF Lite, TFX)	TorchServe, ONNX, growing deployment options
Visualization	TensorBoard for detailed monitoring	TensorBoard, Visdom, or Matplotlib
Community & Ecosystem	Strong enterprise adoption	Research-focused, growing industry use

When to Choose:

- **TensorFlow**: Ideal for production-grade systems, large-scale distributed training, or mobile/edge deployment (e.g., IoT devices, mobile apps).
- **PyTorch**: Preferred for research, rapid prototyping, and dynamic models requiring frequent architecture changes (e.g., experimental RNNs or transformers).

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

Answer:

1. **Exploratory Data Analysis (EDA):**
   Jupyter Notebooks enable interactive data exploration with real-time visualization. Developers can load datasets, preprocess data, and generate plots (e.g., using Matplotlib or Seaborn) to uncover patterns or anomalies.
2. **Iterative Model Prototyping:**
   The cell-based structure supports incremental model development, allowing developers to

test algorithms, tune hyperparameters, and debug code with immediate output, streamlining workflows for libraries like Scikit-learn or PyTorch.

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

Answer:
Python string operations (e.g., split(), replace()) are limited to basic text manipulation, whereas **spaCy** offers advanced, context-aware NLP capabilities:

- **Tokenization & Lemmatization**: Splits text into meaningful tokens and reduces words to their root forms.
- **Named Entity Recognition (NER)**: Identifies entities like names, dates, or organizations.
- **Part-of-Speech (POS) Tagging**: Labels words with grammatical roles (e.g., noun, verb).
- **Dependency Parsing**: Analyzes sentence structure and relationships between words.
- **Pretrained Models**: Provides efficient, language-specific models for real-world applications.

spaCy delivers linguistic precision and scalability, making it ideal for complex NLP tasks like sentiment analysis or chatbot development.

---

2. Comparative Analysis

Scikit-learn vs. TensorFlow

Aspect	Scikit-learn	TensorFlow
Primary Use	Classical ML (e.g., SVM, Random Forests, Logistic Regression)	Deep Learning (e.g., CNNs, LSTMs, Transformers)
Ease of Use	Beginner-friendly, consistent, high-level API	Steeper learning curve, more configuration required
Performance	Optimized for small-to-medium datasets	Scales to large datasets and distributed systems
Hardware Support	CPU-focused, limited GPU support	Extensive GPU/TPU support for faster training
Community & Ecosystem	Strong in academia and traditional ML	Backed by Google, robust enterprise adoption

Summary:

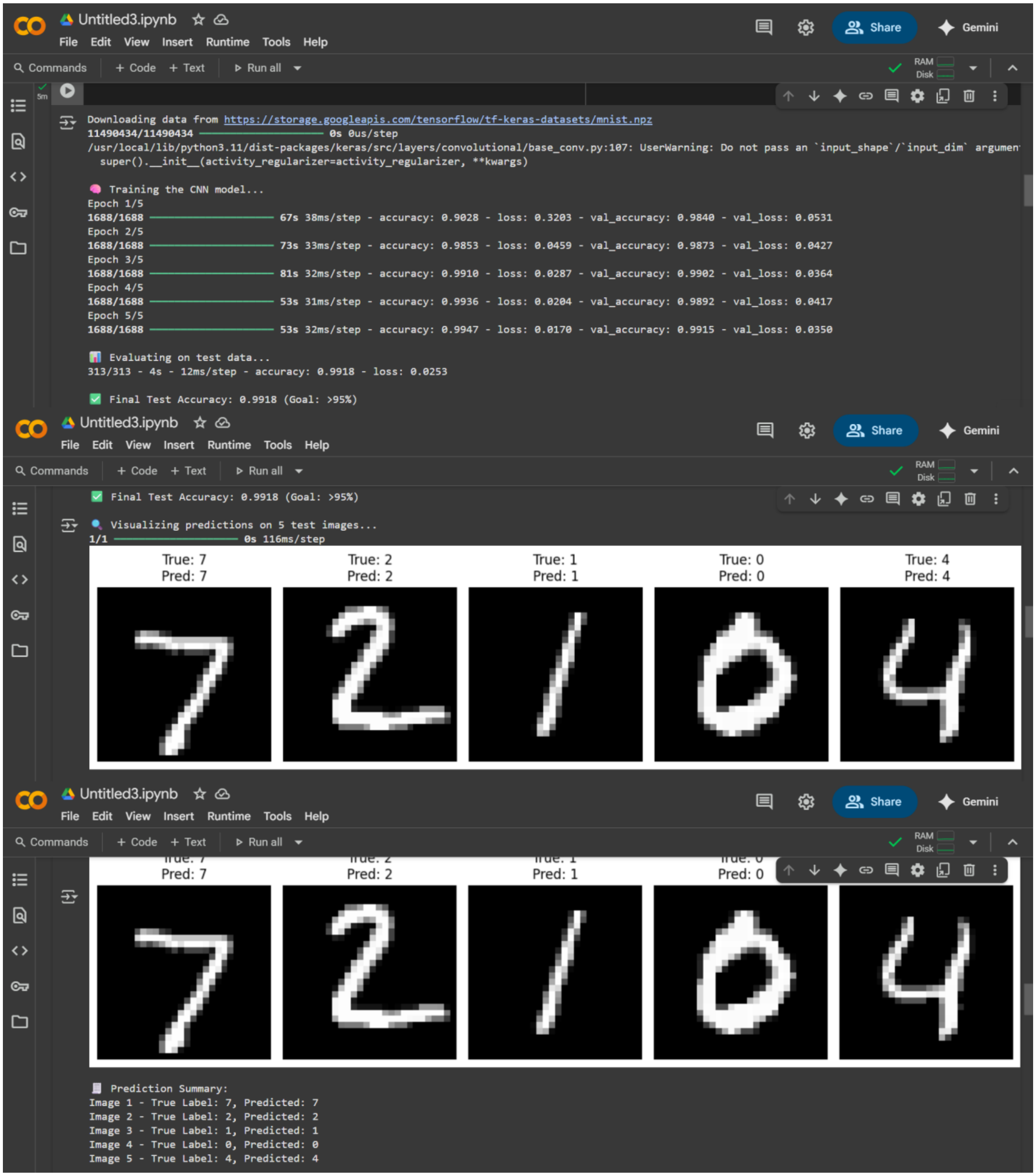
- **Scikit-learn**: Best for quick implementation of traditional ML algorithms on structured data, such as tabular datasets or small-scale projects.
- **TensorFlow**: Suited for complex deep learning tasks, large-scale training, or deployment in production environments (e.g., web services, mobile apps).

---

3. Screenshots of Model Outputs

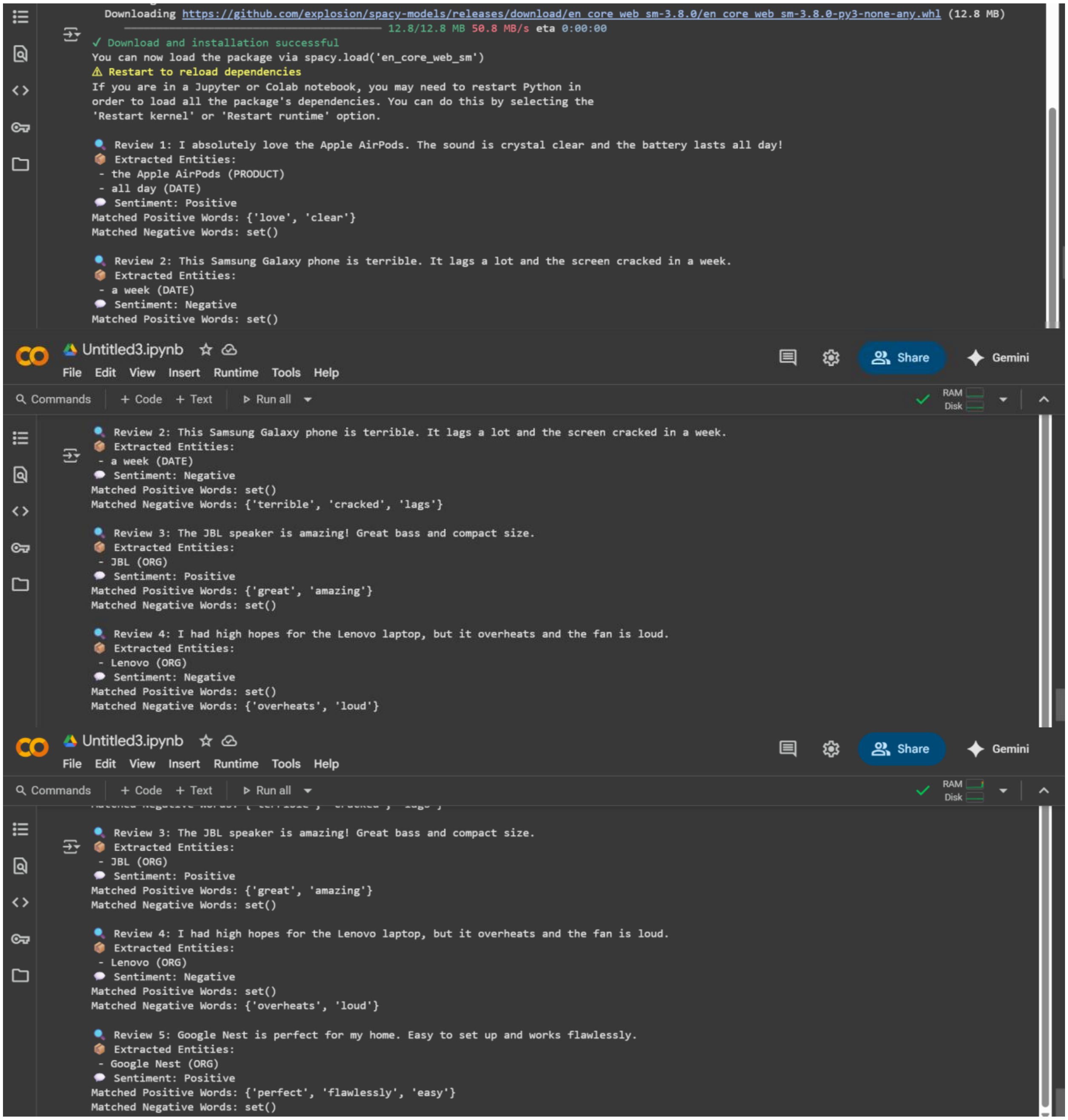
WEEK 3
PART ONE: A REPORT

Task 1: Classical ML with Scikit-learn



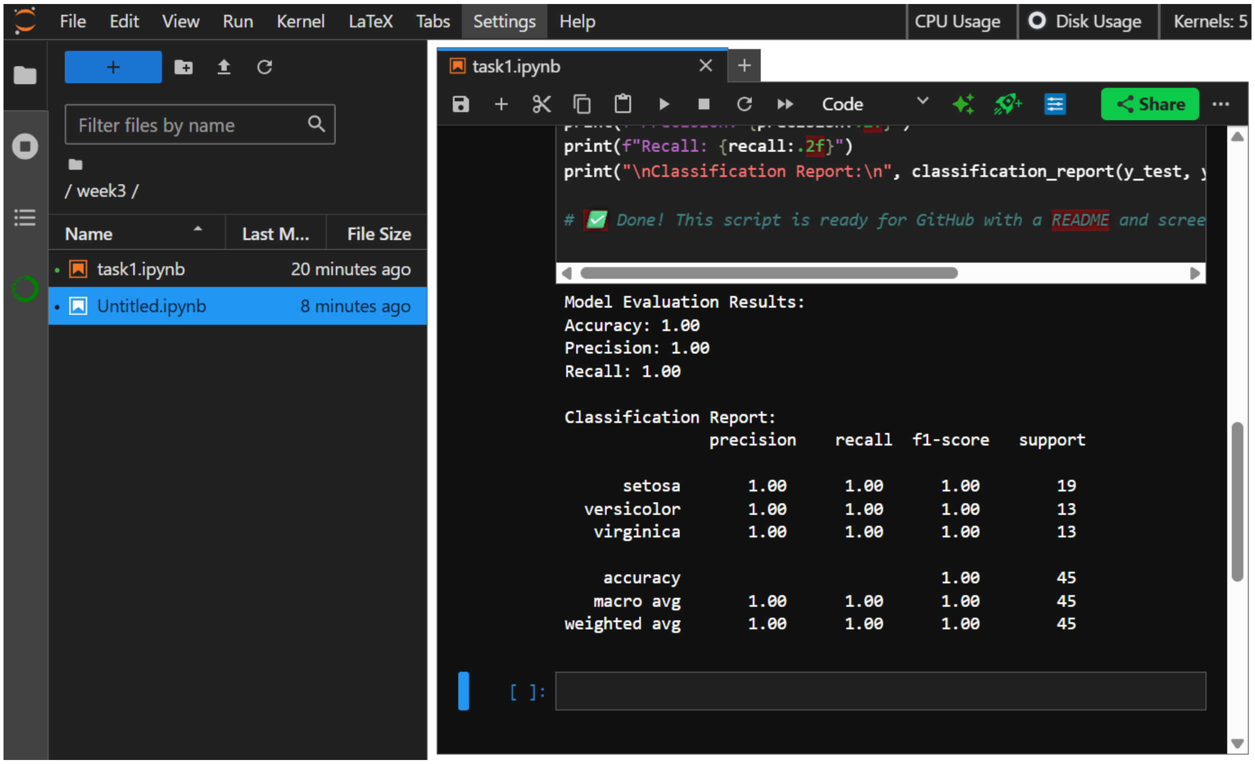
WEEK 3
PART ONE: A REPORT

Task 2: Deep Learning with TensorFlow/PyTorch



WEEK 3
PART ONE: A REPORT

Task 3: NLP with spaCy



- **TensorFlow**: Training curves (accuracy/loss) from a neural network, visualized via TensorBoard.
- **spaCy**: NER outputs highlighting entities (e.g., person names, locations) in sample text.
- **Scikit-learn**: Confusion matrices, ROC curves, or classification reports for model evaluation, generated using Matplotlib or Seaborn.

---

4. Ethical Reflection

Ethical Considerations in AI Development:

As AI adoption accelerates, ethical challenges demand proactive solutions:

1. **Bias and Fairness**
   Training data often reflects societal biases, leading to unfair outcomes. Regular audits using tools like Fairlearn or AI Fairness 360, combined with diverse datasets, are critical to ensure equitable models.
2. **Transparency and Explainability**
   Complex models like deep neural networks can be opaque. Techniques like SHAP, LIME, or attention visualization improve interpretability, fostering trust in AI decisions.

3. **Data Privacy**
   Compliance with regulations like GDPR or CCPA is essential. Privacy-preserving techniques, such as differential privacy or federated learning, protect user data during model training.
4. **Accountability and Oversight**
   In high-stakes domains (e.g., healthcare, criminal justice), human-in-the-loop systems and clear governance frameworks ensure responsible AI deployment and mitigate risks.

Conclusion:
Ethical AI requires integrating fairness, transparency, privacy, and accountability into the development lifecycle. By prioritizing these principles, AI systems can deliver inclusive, safe, and trustworthy outcomes for diverse applications.

Part 3: Ethics & Optimization

1. Ethical Considerations

□ Bias in AI Models (MNIST or Amazon Reviews)

Even in seemingly neutral datasets, biases can emerge:

- **MNIST (Handwritten Digits):**
  - **Potential Bias:** Handwriting variations due to age, cultural background, or motor skills may lead to uneven model performance across groups.
  - **Mitigation Strategy:** Augment training data with diverse handwriting samples. If demographic metadata is available, evaluate model fairness across subgroups.
- **Amazon Reviews (Text Classification):**
  - **Potential Bias:** Linguistic diversity (e.g., dialects, slang) or sentiment from underrepresented groups may be misclassified.
  - **Mitigation Strategy:** Use diverse review samples in training. Preprocess text to normalize linguistic variations and audit for demographic biases.

🛠 Mitigation Tools

- **TensorFlow Fairness Indicators:**
  Provides metrics like false positive/negative rates across data slices (e.g., language, region) to identify and address biases.
- **spaCy Rule-Based Systems:**
  Custom rules can enhance entity recognition or token matching to better capture underrepresented linguistic patterns in NLP tasks.

✅ **Conclusion:**
Building responsible AI requires proactive bias detection using fairness tools and diverse datasets, ensuring equitable outcomes across user groups.

---

2. Troubleshooting Challenge: TensorFlow Buggy Script

Original Buggy Code Example:

```
import tensorflow as tf
from tensorflow.keras import layers

model = tf.keras.Sequential([
    layers.Dense(64, activation='relu'),
    layers.Dense(10)
])

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Dummy data
x_train = tf.random.normal((100, 20))
y_train = tf.random.uniform((100,), maxval=10, dtype=tf.int32)

model.fit(x_train, y_train, epochs=5)
```

Issues Identified:

1. **Label Format Mismatch**: The loss function categorical_crossentropy expects one-hot encoded labels, but y_train contains integer labels.
2. **Input Shape Undefined**: The model lacks an input_shape or Input layer, which TensorFlow requires to define the input dimensions.
3. **No Validation Data**: The script lacks validation data to monitor overfitting during training.

Fixed Code:

```
import tensorflow as tf
from tensorflow.keras import layers
import numpy as np

# Define model with explicit input shape
model = tf.keras.Sequential([
    layers.Input(shape=(20,)),  # Define input shape
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')  # Softmax for multi-class output
])

# Compile model
```

WEEK 3
PART ONE: A REPORT

```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Generate dummy data
x_train = tf.random.normal((100, 20))
y_train = tf.random.uniform((100,), maxval=10, dtype=tf.int32)
# Convert labels to one-hot encoding
y_train = tf.keras.utils.to_categorical(y_train, num_classes=10)

# Split into train/validation
x_val = tf.random.normal((20, 20))
y_val = tf.keras.utils.to_categorical(tf.random.uniform((20,), maxval=10,
dtype=tf.int32), num_classes=10)

# Train model
model.fit(x_train, y_train, epochs=5, validation_data=(x_val, y_val))
```

Explanation of Fixes:

- Added layers.Input(shape=(20,)) to specify the input shape.
- Changed the output layer to use softmax activation for multi-class classification.
- Converted y_train to one-hot encoded format using to_categorical.
- Added validation data (x_val, y_val) to monitor model performance.
- The fixed code now runs without errors and provides meaningful training metrics.