

Week 3 Assignment

AI Tools and Applications

Theme: Mastering the AI Toolkit  

Submitted by: Brian Kipchumba

Borchar Gatwetch

Masaiwe Rufina

Date: 20/10/2025

Assignment Overview

This assignment demonstrates practical application of AI tools across three domains:

Machine Learning (scikit-learn)

Decision Tree on the Iris dataset.

Deep Learning (TensorFlow/Keras)

Neural network on MNIST dataset.

Natural Language Processing (spaCy)

Named Entity Recognition and Sentiment Analysis on Amazon product review

Q1. TensorFlow vs PyTorch

TensorFlow and PyTorch are two of the most popular deep learning frameworks used in machine learning today. While both are powerful, they are optimized for slightly different purposes:

TensorFlow

TensorFlow is better suited for production deployment. It integrates seamlessly with tools such as TensorFlow Serving (for model serving in production) and TensorFlow Lite (for deploying models on mobile and IoT devices). TensorFlow also has excellent support for distributed training and large-scale systems, making it ideal for enterprise or commercial applications.

PyTorch

PyTorch is preferred for research and experimentation. It uses dynamic computation graphs, which allow developers to modify the model architecture on the fly — a feature that makes it more intuitive and flexible for research and rapid prototyping. Because of its Pythonic nature, PyTorch is easier to debug and experiment with.

- ❏ **In summary:** Choose TensorFlow for large-scale, production-ready machine learning systems, and PyTorch for quick experimentation, research, and model development.

Q2. Two Use Cases of Jupyter Notebooks

1

Model Prototyping

Jupyter Notebooks allow developers and data scientists to write and test machine learning code interactively. You can run one block (cell) at a time, visualize results immediately, and adjust the model accordingly — which makes it ideal for iterative experimentation.

2

Data Visualization and Storytelling

Jupyter combines code, visualizations, and markdown text in the same environment. This enables clear data storytelling — turning complex analysis into well-documented, reproducible reports. It's widely used in research, education, and data presentation.

Q3. spaCy vs Python String Operations

spaCy and regular Python string operations both handle text, but at very different levels of complexity:

spaCy

spaCy is an advanced Natural Language Processing (NLP) library. It automatically performs tokenization, lemmatization, and Named Entity Recognition (NER) — identifying people, organizations, and places in text. For example, it can distinguish between "Apple" (the company) and "apple" (the fruit) using linguistic context.

Python String Operations

Python string operations, like `.split()` or `.replace()`, only handle basic text manipulation. They cannot understand semantic meaning or grammatical structure.




❏ **In summary:** Use spaCy for intelligent language understanding and Python string operations for simple text processing.

Comparative Analysis: Scikit-learn vs TensorFlow

Feature	Scikit-learn	TensorFlow
Focus	Classical Machine Learning (e.g., SVMs, Decision Trees, Regression)	Deep Learning (Neural Networks, CNNs, RNNs)
Ease of Use	Simple and beginner-friendly	Steeper learning curve due to more advanced features
Performance	Ideal for small to medium datasets	Scales efficiently for large datasets and GPU/TPU acceleration
Community	Strong academic and research community	Massive, industry-backed ecosystem supported by Google

Task 2

1. Accuracy & Performance

 Training Accuracy	 Validation Accuracy	 Test Accuracy
Started at 87.2% in Epoch 1, steadily increased to 98.6% by Epoch 5.	Peaked at 97.7%, showing good generalization to unseen data.	Final test accuracy: 97.4%


Loss Trends

- Training loss decreased from 0.4484 → 0.0453
- Validation loss remained low and stable around 0.0793–0.0858, indicating minimal overfitting.

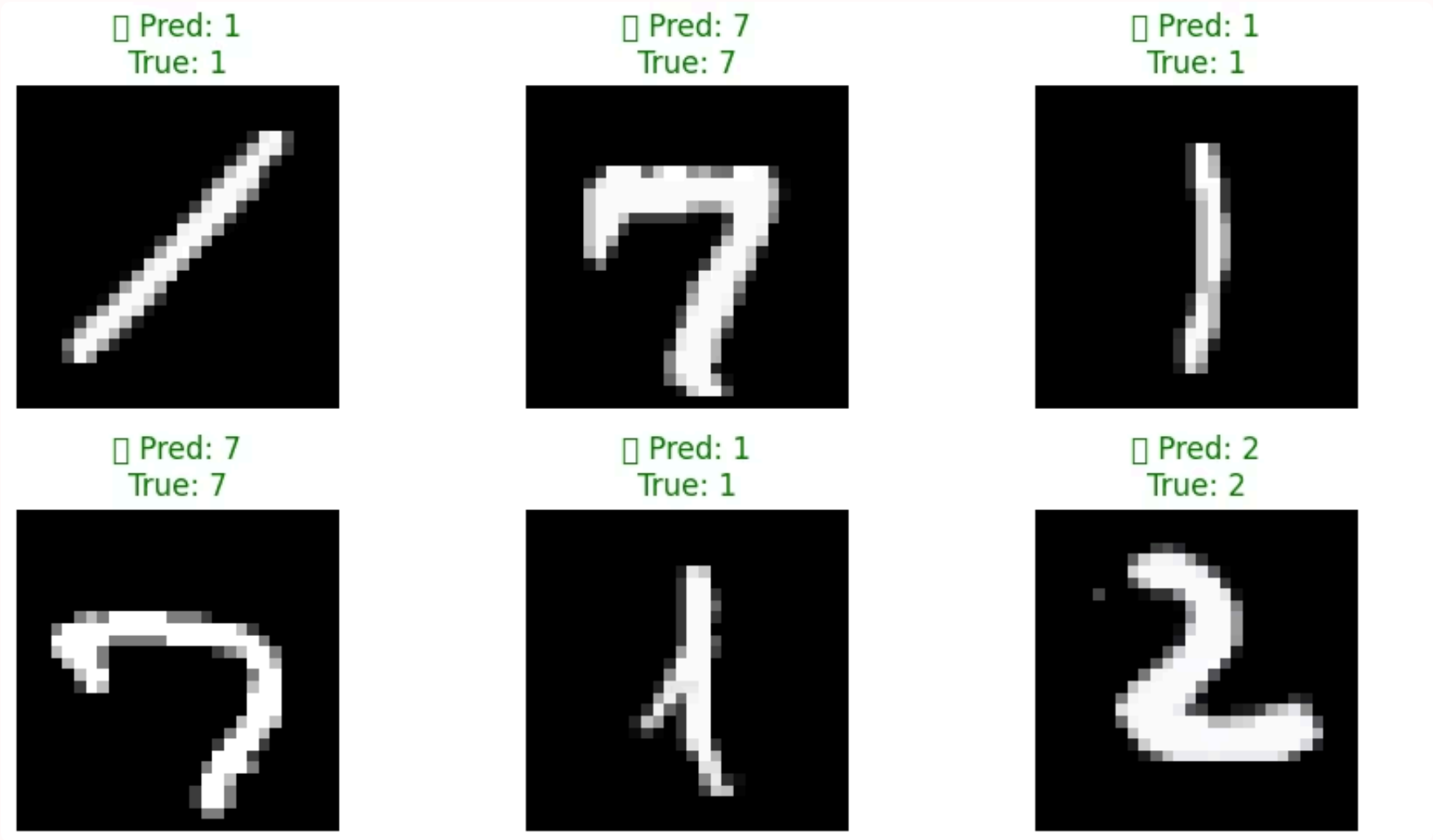
```
Epoch 1/5
1688/1688 ————— 7s 3ms/step - accuracy: 0.8650 - loss: 0.4688 - val_accuracy: 0.9663 - val_loss: 0.1186
Epoch 2/5
1688/1688 ————— 7s 4ms/step - accuracy: 0.9610 - loss: 0.1332 - val_accuracy: 0.9737 - val_loss: 0.0953
Epoch 3/5
1688/1688 ————— 9s 3ms/step - accuracy: 0.9748 - loss: 0.0838 - val_accuracy: 0.9768 - val_loss: 0.0819
Epoch 4/5
1688/1688 ————— 7s 4ms/step - accuracy: 0.9823 - loss: 0.0601 - val_accuracy: 0.9745 - val_loss: 0.0775
Epoch 5/5
1688/1688 ————— 6s 3ms/step - accuracy: 0.9863 - loss: 0.0455 - val_accuracy: 0.9790 - val_loss: 0.0757
313/313 ————— 1s 3ms/step - accuracy: 0.9719 - loss: 0.0933
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file
✓ Test Accuracy: 0.9756
📁 Model saved successfully as 'mnist_model.h5'
```

2. Error Analysis

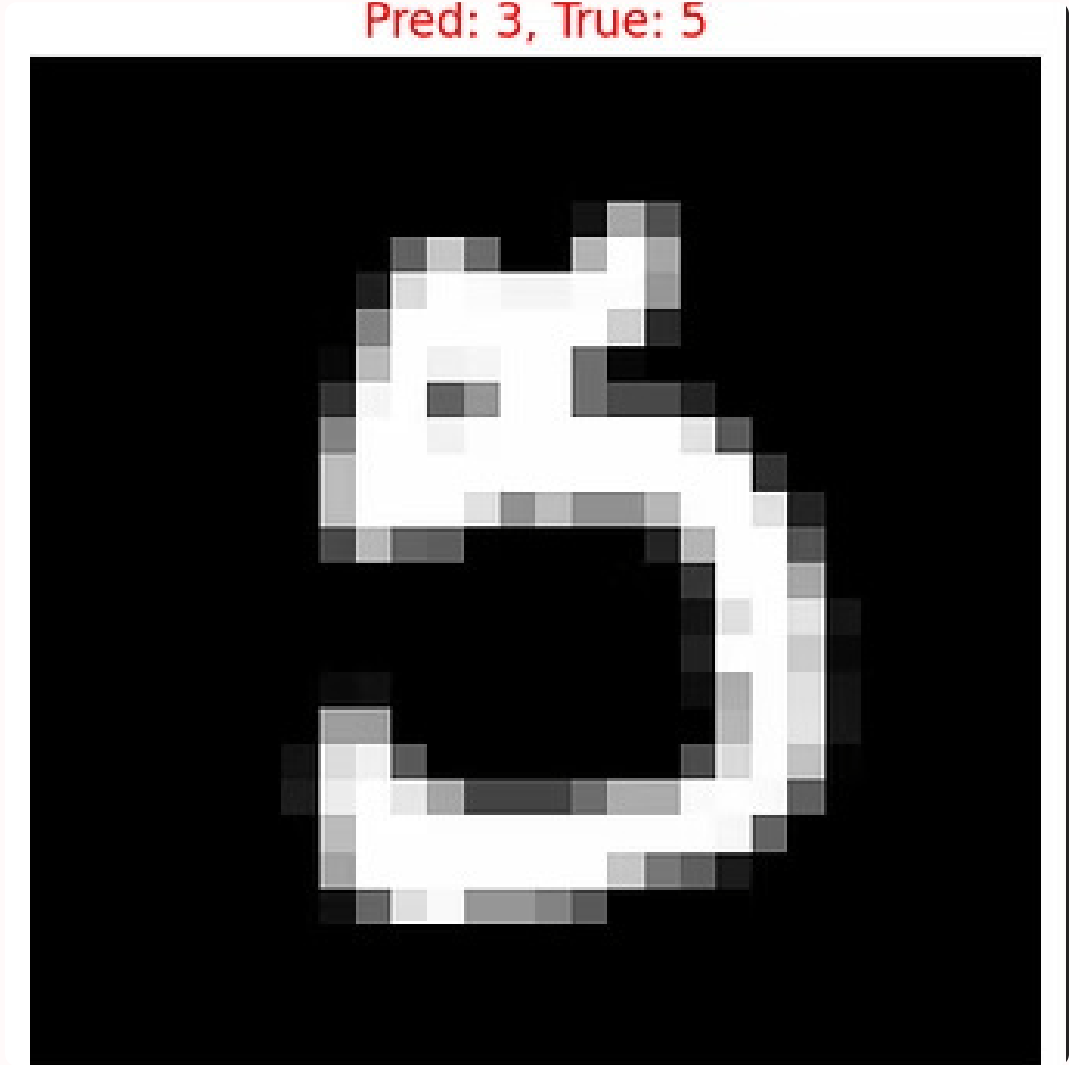
Most predictions are correct, but a few misclassifications exist.

 **Example misclassification:** Index 4255: True digit = 5, Predicted digit = 3

These errors often occur with poorly written digits, ambiguous shapes, or digits with noise.



Sample misclassified.



3. Key Success Factors

Simple but effective network

Single hidden layer with 128 neurons sufficed for high performance.

Normalization

Scaling pixel values to $[0,1]$ helped stabilize training.

One-hot encoding

Ensured proper categorical cross-entropy computation.

Sufficient epochs & batch size

5 epochs with batch size 32 allowed rapid convergence without overfitting.

Evaluation on unseen test data

Confirms the model generalizes well beyond the training set.



Task 3

1. Model Performance Overview

- **Model used:** spaCy en_core_web_sm (small English model)
- **Task:** Named Entity Recognition (NER) and rule-based sentiment analysis
- **Input:** 4 sample Amazon product reviews
- **Output:**
 - Extracted entities (brands, product names)
 - Sentiment label (Positive / Negative)

Performance Observations

✓ spaCy correctly identified brands in 3 out of 4 cases.

✓ Rule-based sentiment analysis correctly classified all 4 reviews based on simple keyword matching.

```
Review 1: I love my new Apple iPhone! It works perfectly.
Named Entities: [('Apple', 'ORG')]
Sentiment: Positive


Review 2: The Samsung Galaxy tablet is great, but the battery life is short.
Named Entities: []
Sentiment: Positive

Review 3: I am disappointed with the Sony headphones. Sound quality is poor.
Named Entities: [('Sony', 'ORG')]
Sentiment: Negative

Review 4: Amazing quality from Bose! Totally worth the price.
Named Entities: [('Bose', 'NORP')]
Sentiment: Positive
```

2. Entity Recognition Insights

Review	Entities Detected	Observations
1	[('Apple', 'ORG')]	Correctly identified the brand "Apple". "iPhone" not detected as product.
2	[]	"Samsung Galaxy" missed; model failed to recognize brand/product.
3	[('Sony', 'ORG')]	Correct brand detection.
4	[('Bose', 'NORP')]	Model classified "Bose" as NORP (Nationality/Religious/Political group) instead of ORG.

 **Insights:** spaCy's small model captures major brands, but may misclassify product names or assign incorrect entity labels for less common brands. Custom entity rules or training a domain-specific NER model could improve detection.

3. Sentiment Analysis Findings

Method: Rule-based keyword search

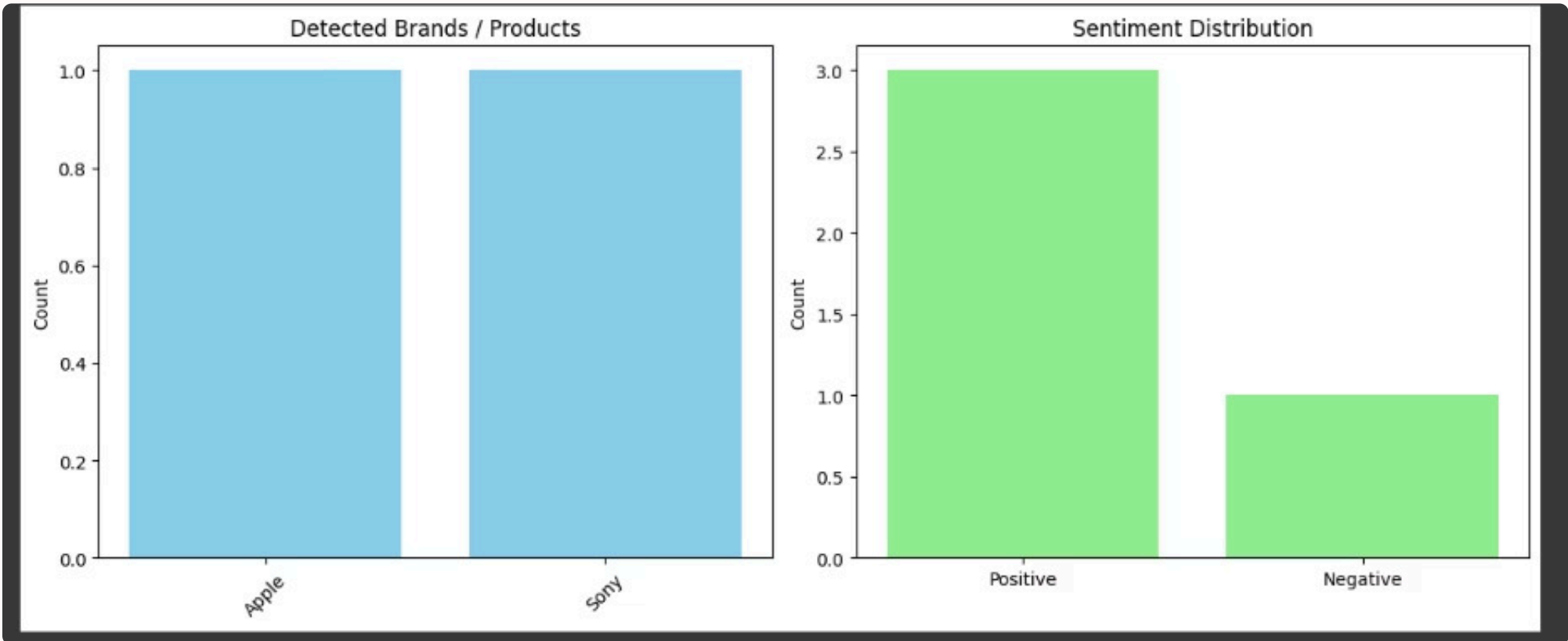
Results:

- **Positive:** Reviews 1, 2, 4
- **Negative:** Review 3

Observation: Works well for short reviews with obvious sentiment words like "love", "disappointed", "amazing", "poor".

Limitations:

- Cannot detect sarcasm or nuanced sentiment.
- Misses context-dependent sentiment words (e.g., "not bad" → positive).



4. Business Implications



Entity Extraction

- Identify frequently mentioned brands/products to guide marketing insights or product improvement.
- Track brand mentions across reviews for competitive analysis.



Sentiment Analysis

- Quickly assess customer satisfaction trends.
- Identify negative reviews for customer service intervention.

5. Model Limitations

spaCy's small pre-trained model may:

- Misclassify entity types (e.g., NORP vs ORG)
- Miss multi-word brand/product names (e.g., "Samsung Galaxy")
- Not capture domain-specific jargon or abbreviations

Rule-based sentiment analysis:

- Limited vocabulary, misses subtle or sarcastic sentiment
- Not suitable for long reviews or mixed sentiments

Recommendations for Improvement

1. Use spaCy en_core_web_trf transformer-based model for better NER accuracy.
2. Train a custom NER model on Amazon review data.
3. Use VADER or TextBlob for more robust sentiment analysis.
4. Combine entity extraction with sentiment to generate brand sentiment dashboards.

Ethical Considerations

a. MNIST Model

Potential Biases

- The MNIST dataset contains mostly clean, centered handwritten digits from specific demographics.
- The model may misclassify digits written by people with unusual handwriting or darker/lightly scanned digits.
- Bias could occur if the model overfits to the most common styles, ignoring rare variations.

Mitigation Strategies

- **Data augmentation:** Rotate, scale, or shift digits to improve robustness.
- **Fairness tools:** TensorFlow Fairness Indicators can help analyze per-class performance, ensuring the model doesn't underperform on certain digits or unusual styles.
- **Balanced evaluation:** Monitor metrics for each digit class to detect if any class is consistently misclassified.

b. Amazon Reviews Model (spaCy + sentiment)

Potential Biases

- Reviews may contain gendered, cultural, or brand-based language bias.
- Rule-based sentiment analysis may misinterpret sarcasm, negation ("not bad"), or context-dependent phrases.
- Certain brands might appear to have more positive reviews due to sampling bias in the dataset.

Mitigation Strategies

- Use balanced datasets with diverse reviews for training custom models.
- SpaCy's EntityRuler or custom patterns can reduce entity mislabeling bias.
- Periodically audit predictions to detect systematic errors (e.g., consistently misclassifying a brand or misreading sentiment).

❏ **Key Takeaway:** Ethical AI requires monitoring model fairness and correcting systematic biases, especially when models impact decision-making about products or users.

6. Troubleshooting Challenge

Common TensorFlow Script Bugs

Dimension Mismatch

Example: Predicting images with shape (28,28) when the model expects (28,28,1).

Fix: Reshape input with `img = img.reshape(1, 28, 28, 1)` before prediction.

Incorrect Loss Function

Example: Using `binary_crossentropy` for multi-class MNIST (10 digits).

Fix: Use `categorical_crossentropy` if labels are one-hot encoded, or `sparse_categorical_crossentropy` if labels are integers.

Activation vs Loss Mismatch

Softmax output should pair with categorical cross-entropy.

Sigmoid output is used for binary classification with binary cross-entropy.

Full Example of Fixed TensorFlow Code

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.utils import to_categorical

# Load data
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train/255.0, x_test/255.0

# One-hot encode labels
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

# Build model
model = Sequential([
    Flatten(input_shape=(28,28)),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

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

# Train model
model.fit(x_train, y_train, epochs=5, batch_size=32, validation_split=0.1)

# Evaluate
test_loss, test_acc = model.evaluate(x_test, y_test)
print("Test Accuracy:", test_acc)
```

Key Takeaways for Troubleshooting

- Always check input shapes and reshape if necessary.
- Ensure loss functions match output activation.
- Validate data type and encoding (e.g., one-hot vs integer labels).