

# PART 1

## 1.. Short Answer Questions

### QN1.

They both help you build smart computer programs, but they work a little differently.

TensorFlow: with TensorFlow first have to draw out all your plans for what you're building, step-by-step, before you even start, you must have a detailed plan. Once the plan is done, the computer can build it fast,

especially if it's a huge project. This is great for when you want to make something super strong and reliable that will be used by lots of people, like an app on your phone.

It's like an architect who draws every detail before construction begins.

PyTorch: PyTorch is more like a building kit where you can start building right away. You put one piece, then another, and you can change your mind or fix mistakes as you go.

This makes it really easy to try out new ideas and experiment quickly. It's like playing with LEGOs; you just start building and see what works best, making adjustments as you go.

This is very popular for researchers and people trying out new AI ideas.

When to choose one:

You'd lean towards PyTorch if you're exploring new ideas, learning, or need to fix things easily as you go. It's flexible and quick for experiments.

You'd pick TensorFlow if you're making a finished product that needs to be super fast and reliable for many users, like a big company's app or a system used globally.

### QN2.

#### JUPYTER NOTEBOOK.

it's a space where you can actually run little bits of code snippets, see the results immediately, and write down your thoughts "markdowns", all on the same page!

Here are two ways we use them:

Understanding your data- Imagine you have a big pile of photos, and you want to teach a computer to find all the dogs. In a Jupyter Notebook, you can look at a few photos, count how many dogs there are, check if any photos are blurry,

and clean up anything that's messy. You run a small piece of code, see the picture, then run another piece to count, and so on. It's perfect for understanding your data step-by-step before you use it to train your AI. Building and Testing AI Models : Once your data is ready, you use the Jupyter Notebook to actually build your AI program. You can write a small piece of code to teach the computer to recognize a dog,

then immediately see if it worked. If not, you can change a few lines of code right there and try again. This "try, see, fix, try again" process is super fast and efficient in a Jupyter Notebook,

allowing you to experiment with different AI ideas until you find the best one.

### **QN3.**

spaCy significantly enhances NLP tasks compared to basic Python string operations by providing a robust, efficient, and pre-trained framework for linguistic processing, allowing for much deeper and more accurate analysis of text.

Here's how:

-When you use basic Python string operations, it's like just looking at the letters in a word. If you see "run", you know it has the letters 'r', 'u', 'n'. You can count them or change 'n' to 'm' to make "rum". That's all.

But spaCy is like your smart detective friend. If you give spaCy the word "run", it knows:

It's a verb (something you do).

It can be used to mean "running very fast" or "the machine is running."

It can even figure out that "running" is related to "run."

It can spot that "New York" is a place (a "named entity"), not just two separate words.

So, while basic string operations just help you count letters or swap them, spaCy helps the computer understand the true meaning behind sentences, just like your smart detective friend understands what people are really trying to say!

This makes it super helpful for computers to read stories, answer questions, or even talk to you!.

## **2. Comparative Analysis**

-Target applications (e.g., classical ML vs. deep learning).

Scikit-learn: This is your fantastic set of "classical machine learning" tools.

What it's for: Imagine you want to teach a computer to do things that are smart, but maybe not super complex, like:

Predicting house prices: Based on how many rooms a house has, its size, etc.

Sorting emails: Telling if an email is important or junk (spam).

Finding groups in data: Like grouping customers based on their shopping habits.

It's like a skilled carpenter who builds amazing things with standard, well-understood methods. These "classical" methods are often great when you don't have a massive amount of data.

TensorFlow: This is your "deep learning" robot factory.

What it's for: TensorFlow is designed for much bigger, more complex brain-like structures called "neural networks" (which is what deep learning uses).

Image Recognition: Teaching a computer to "see" and understand what's in a picture (e.g., identifying different types of animals, people's faces).

Natural Language Understanding: Teaching a computer to truly understand human language, not just words, but the meaning behind sentences (e.g., chatbots, language translation).

Generating new content: Like making AI write stories or create realistic images.

It's like an advanced engineer building complex robots that can "think" and "learn" in ways that are closer to how our own brains work, especially when you have mountains of data.

-- Ease of use for beginners.

Scikit-learn: This is generally considered easier for beginners to pick up.

Why: Its tools are often straightforward to use. You usually just plug in your data, pick a tool (an "algorithm"), and it does the job. It's like having a well-labeled toolkit where each tool does one clear thing,

and you can see how it works. You can get results quickly without diving into very complex concepts.

TensorFlow: This can be more challenging for beginners.

Why: Because it's designed for those "deep learning" brain-like structures, it requires understanding a bit more about how those complex "brains" are put together. It's like learning to operate a sophisticated factory;

there are more buttons, more settings, and you need to understand the underlying machinery. While TensorFlow has improved its beginner-friendliness with things like Keras (which simplifies building models),

it still requires a bigger leap in understanding compared to Scikit-learn.

-- Community support

Scikit-learn: Has excellent community support, especially for classical ML tasks.

What this means: There are tons of helpful guides, tutorials, and examples online. If you get stuck, many people have probably faced the same problem, and you can easily find answers on forums like Stack Overflow.

It's been around for a long time and is widely adopted by data scientists.

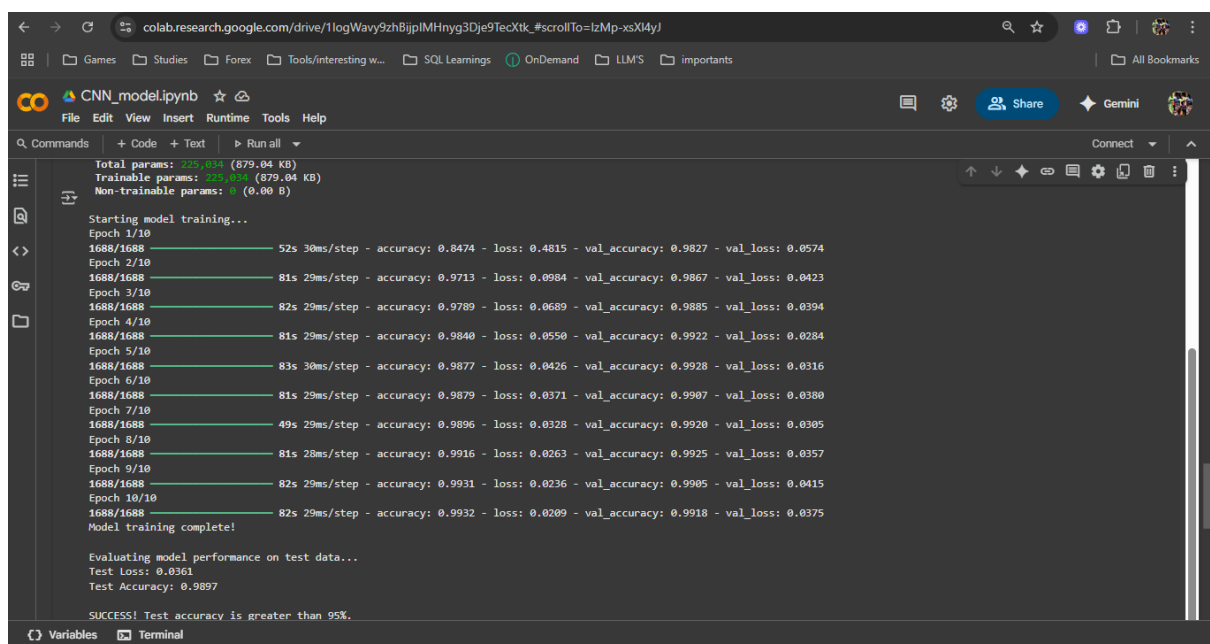
TensorFlow: Has massive community support, especially for deep learning.

What this means: Because it's backed by Google and used by researchers and companies worldwide, there's an enormous amount of resources, documentation, tutorials, and a very active community. If you're doing deep learning,

you'll find a vast network of people and resources to help you. It's always evolving, so the community is very active in sharing new developments.

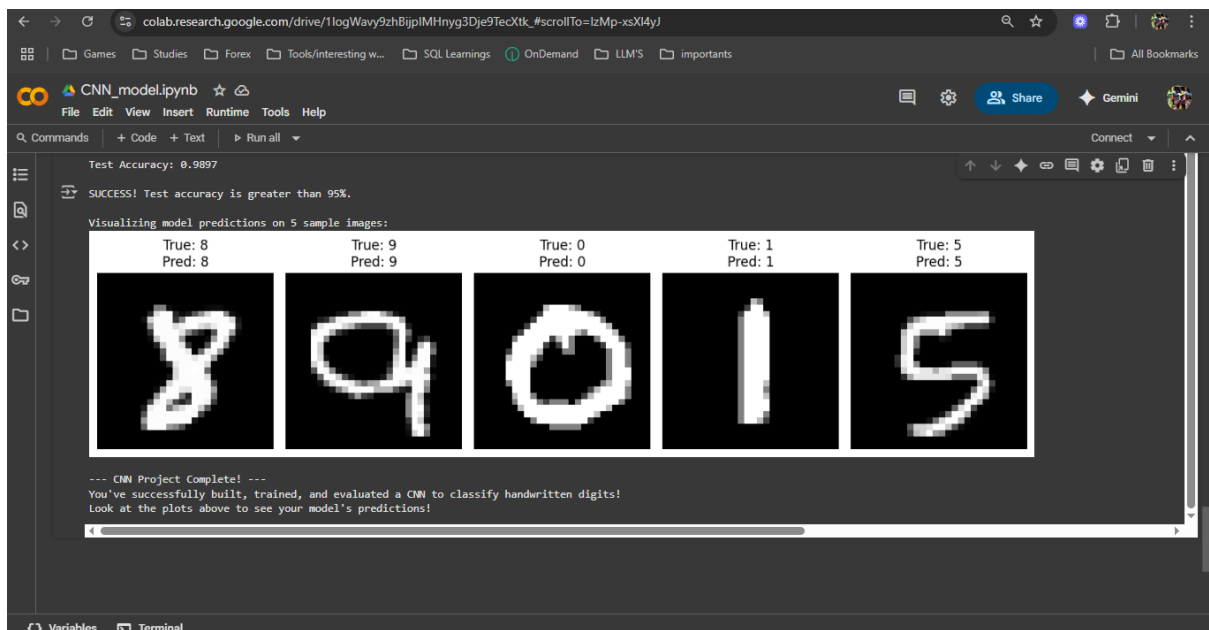
## CNN MODEL OUTPUTS

### Training

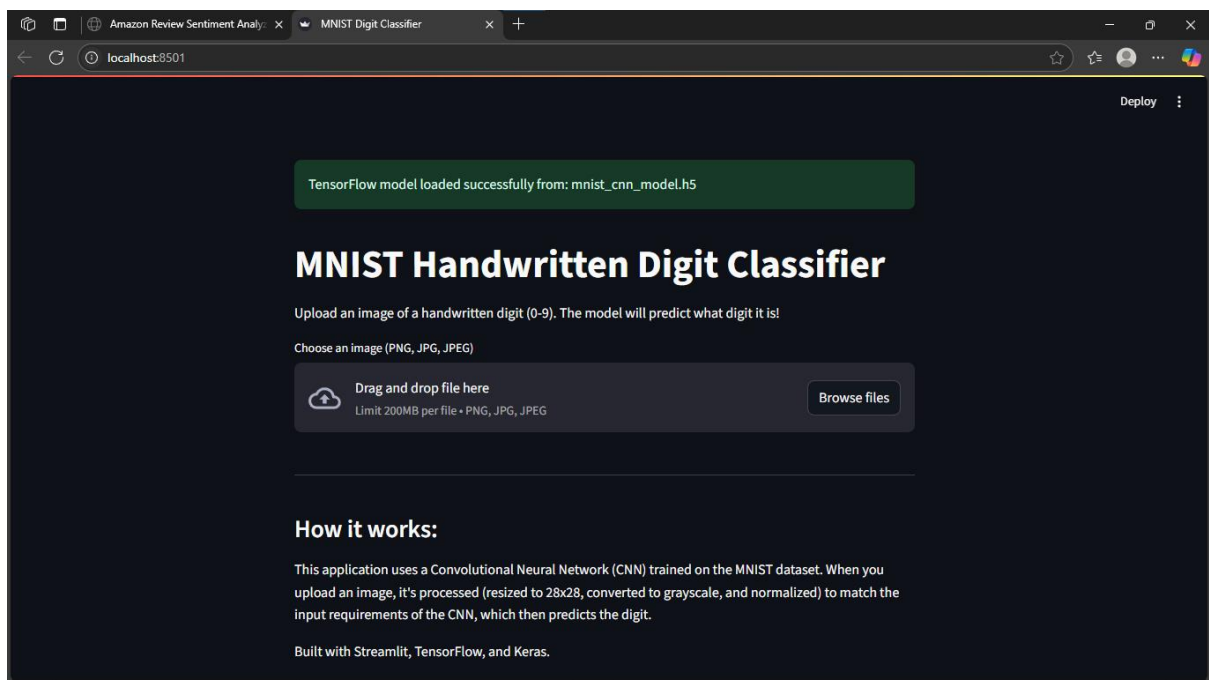


```
colab.research.google.com/drive/1logWavy9zhBijpIMHnyg3Dje9TecXtk_#scrollTo=IzMp-xsXl4y/
CNN_model.ipynb
File Edit View Insert Runtime Tools Help
+ Code + Text ▶ Run all
Total params: 225,044 (879.04 KB)
Trainable params: 225,044 (879.04 KB)
Non-trainable params: 0 (0.00 B)
Starting model training...
Epoch 1/10
1688/1688 — 52s 30ms/step - accuracy: 0.8474 - loss: 0.4815 - val_accuracy: 0.9827 - val_loss: 0.0574
Epoch 2/10
1688/1688 — 81s 29ms/step - accuracy: 0.9713 - loss: 0.0984 - val_accuracy: 0.9867 - val_loss: 0.0423
Epoch 3/10
1688/1688 — 82s 29ms/step - accuracy: 0.9789 - loss: 0.0689 - val_accuracy: 0.9885 - val_loss: 0.0394
Epoch 4/10
1688/1688 — 81s 29ms/step - accuracy: 0.9840 - loss: 0.0550 - val_accuracy: 0.9922 - val_loss: 0.0284
Epoch 5/10
1688/1688 — 83s 30ms/step - accuracy: 0.9877 - loss: 0.0426 - val_accuracy: 0.9928 - val_loss: 0.0316
Epoch 6/10
1688/1688 — 81s 29ms/step - accuracy: 0.9879 - loss: 0.0371 - val_accuracy: 0.9907 - val_loss: 0.0380
Epoch 7/10
1688/1688 — 49s 29ms/step - accuracy: 0.9896 - loss: 0.0328 - val_accuracy: 0.9920 - val_loss: 0.0305
Epoch 8/10
1688/1688 — 81s 28ms/step - accuracy: 0.9916 - loss: 0.0263 - val_accuracy: 0.9925 - val_loss: 0.0357
Epoch 9/10
1688/1688 — 82s 29ms/step - accuracy: 0.9931 - loss: 0.0236 - val_accuracy: 0.9905 - val_loss: 0.0415
Epoch 10/10
1688/1688 — 82s 29ms/step - accuracy: 0.9932 - loss: 0.0209 - val_accuracy: 0.9918 - val_loss: 0.0375
Model training complete!
Evaluating model performance on test data...
Test Loss: 0.0361
Test Accuracy: 0.9897
SUCCESS! Test accuracy is greater than 95%.
```

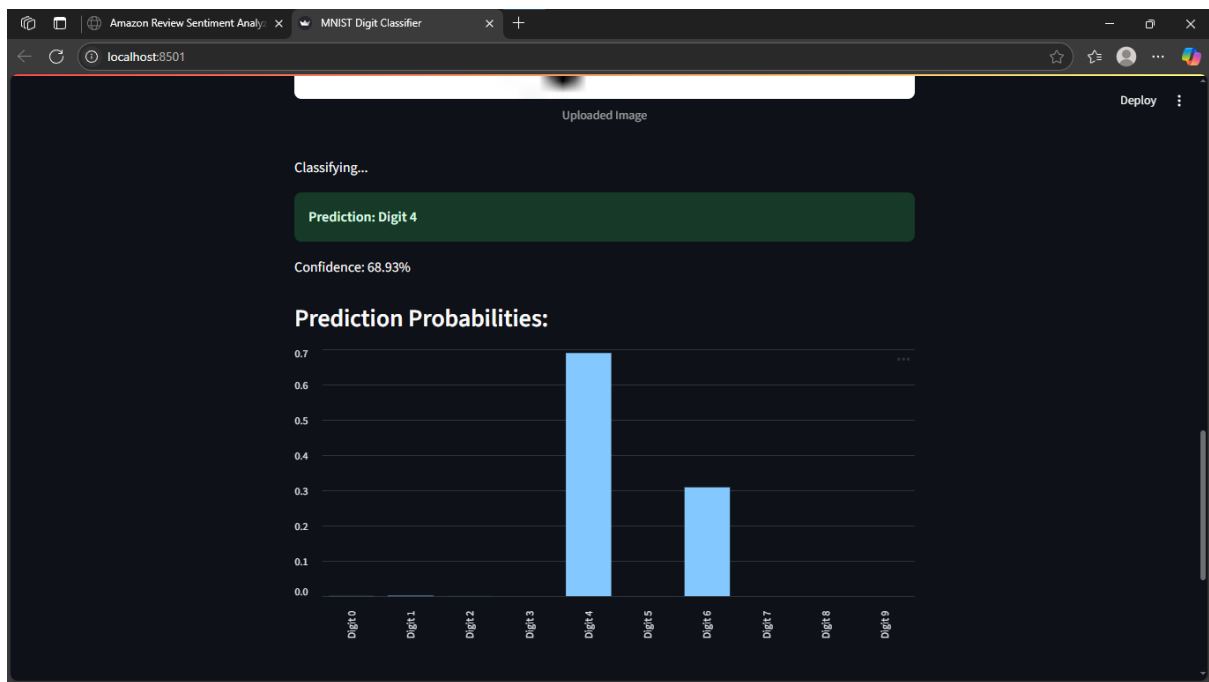
### Visual Output 5 sample



## UI streamlit



## Prediction of streamlit app



## AMAZON SENTIMENT ANALYSIS screenshot of output

```
colab.research.google.com/drive/18JzUJTpSo-gaq2kN3bb0leiCsO9m1WKD#scrollTo=PQyf-S7YxwJ

spacy.ipynb
File Edit View Insert Runtime Tools Help
+ Code + Text Run all
Q Commands

# Recently bought the new AcmeUltra laptop by Acme Inc., and its performance is outstanding.
# The battery life, screen quality, and build are all top-notch. However, the customer support could be better.
"""

# Process the text with spacy
doc = nlp(review_text)

# Extract and print named entities (e.g., product names, organizations)
print("Named Entities:")
for ent in doc.ents:
    print(f"{ent.text} - {ent.label}")

Named Entities:
AcmeUltra Laptop - PRODUCT
Acme Inc. - ORG

# Rule-based sentiment analysis: a simple approach based on keyword presence
positive_keywords = ["outstanding", "top-notch", "excellent", "good"]
negative_keywords = ["bad", "poor", "could be better", "disappointing"]

# Lowercase conversion for matching
review_lower = review_text.lower()
sentiment = "Positive" if any(word in review_lower for word in positive_keywords) else "Negative"
print(f"\nSentiment: {sentiment}")

Sentiment: Positive
```

## PART 3: ETHICS AND OPTIMIZATION

The MNIST model can have biases because of different handwriting styles, unclear digit shapes, or small differences in how many of each number are in the dataset. This can cause the model to work better on some digits than others, or on digits written in certain ways.

For the Amazon Reviews model, bias may come from the way people write reviews—some users may use more positive or negative words, or speak differently based on location, gender, or age. If the training data isn't balanced, the model might favor one group's writing style over another.

Tools like TensorFlow Fairness Indicators can help by checking how well the model performs for different groups or data slices. Even though MNIST doesn't include labels like age or gender, we can still test how the model does on different styles of writing.

spaCy's rule-based systems, even though made for text, can inspire rules for images or reviews. These rules could help find uncommon patterns or styles and guide data improvement. This could include adding more examples of underrepresented styles to make the model fairer and more balanced.

In short, spotting and fixing these small issues can help make our models more fair and accurate.