Name: Shayan Umar

Designation: AI intern

Department: Artificial Intelligence

Date: 7-25-2025

Table of Contents

1. Theoretical Understanding:

1.1 Neural Networks:

1. Neural Networks (ANNs)

A **neural network** is a computer system inspired by the **human brain**. It's made up of layers of **"neurons" or "nodes"** that try to learn patterns from data.

Think of it like a smart calculator that can **learn to recognize handwriting, detect diseases, or classify images** by being shown lots of examples.

---

2. Nodes and Hidden Layers

- **Node** (Neuron): A small computing unit that:
    - Takes input (like a number)
    - Multiplies it by a weight
    - Adds a bias
    - Applies an **activation function**
    - Sends the result to the next layer
- **Layers**:
    - **Input Layer**: Where data enters (e.g., pixel values of an image)
    - **Hidden Layers**: Layers between input and output — this is where the **real learning happens**
    - **Output Layer**: Produces the final result (like "cat" or "dog")

---

3. Activation Functions

Activation functions decide whether a neuron should "fire" or not — like a decision-making switch.

Here are some **popular ones**:

Function	Output Range	Use Case	Key Idea
Sigmoid	0 to 1	Binary classification	Converts input to probability-like output
Tanh	-1 to 1	When output should be centered	Better than sigmoid in many cases
ReLU (Rectified Linear Unit)	0 to ∞	Deep networks	Keeps positive values, turns negatives to 0

---

4. Backpropagation

Backpropagation is **how a neural network learns**.

1. It makes a **prediction**.
2. It compares the prediction to the **true answer** (using a loss function).
3. It **adjusts the weights** backward through the network to improve.

This is like:

Guessing an answer → Seeing how wrong you were → Fixing your thinking for next time.

---

5. Vanishing and Exploding Gradients

These are **training problems** in deep networks.

Problem	What Happens	Why It's Bad
Vanishing Gradient	Gradients shrink too much	Early layers stop learning
Exploding Gradient	Gradients become too large	Model becomes unstable (outputs NaN or huge values)

They happen during backpropagation, especially with **many layers**.

---

6. Multiclass Classification

This is when your model must choose **one class from many** (e.g., cat, dog, horse, elephant).

There are **two ways** to handle this in the output layer:

a. Sigmoid Activation (One-vs-All)

- Apply **sigmoid** to each output neuron **independently**.
- Good for **multi-label problems** (e.g., an image can be both "dog" and "cute").
- **Outputs don't sum to 1**.

b. Softmax Activation (True Multi-Class)

- Use **softmax** so all outputs **sum to 1**.
- Gives you **relative probabilities**.
- Best for **mutually exclusive classes** (e.g., an animal can't be both cat and dog at once).

1.2 Embeddings:

1. What Are Embeddings?

Imagine you want a computer to understand the word **"king"**. But computers don't understand words — they understand **numbers**.

So, **embeddings** are a way of turning **words, images, or any data** into a set of **numbers (vectors)** that capture their **meaning** or **features**.

An **embedding** is a numeric representation of an item (like a word) in a way that similar items are close together.

Example:

- **"king"** → [0.52, 1.1, -0.3, ..., 0.8]
- **"queen"** → [0.49, 1.2, -0.4, ..., 0.7]
  They're close because they **mean similar things**.

---

2. What Is an Embedding Space?

Think of an **embedding space** as a **map** where each word or item is a **dot (vector)**. The position of the dot shows:

- What it means
- How it relates to other dots

For example:

- "king", "queen", "prince", "princess" might be close to each other.
- "dog", "cat" will be in another region of the space.
- "banana", "apple" will be in another.

The **embedding space** is the **vector space** where all these embeddings live.

---

3. What Is a Vector Space?

A **vector space** is a mathematical space where:

- Each item is represented by a **vector** (a list of numbers like [1.3, 0.5, -0.9])
- You can calculate **distance**, **similarity**, and **direction**

In the context of embeddings:

- It allows you to say things like:
  - **"king" - "man" + "woman" ≈ "queen"**
    (this works in good embeddings like Word2Vec!)

---

4. What Are Static Embeddings?

Static embeddings are **fixed** for each word, no matter the context.

Examples:

- **Word2Vec**

- **GloVe**
- **FastText**

The word **“bank”** has **one meaning** whether it’s a **river bank** or a **money bank**.

That’s a limitation — the model can’t adapt to context.

---

5. What Are Dynamic Embeddings?

Dynamic embeddings change **based on the context** of the word.

Examples:

- **BERT**
- **GPT**
- **ELMo**

In a dynamic embedding model:

- “I sat by the river bank.” → “bank” means **nature**
- “I deposited money in the bank.” → “bank” means **finance**

These models generate **different embeddings** for the **same word** depending on the sentence.

Analogy

Imagine words as **locations on a map**:

- Static = each word has **one fixed address**
- Dynamic = the **address changes** depending on what city you’re in (context)

1.3 Large Language Models (LLMs):

What is a Language Model?

A **language model (LM)** is a tool that predicts the next word (or token) in a sentence based on the previous ones. A token can be a word, part of a word (subword), or even a character.

For example, given this sentence:

"When I hear rain on my roof, I _____ in my kitchen."

The model guesses possible completions like:

- **cook soup** (9.4%)
- **warm up a kettle** (5.2%)
- **cower** (3.6%)

The model uses probabilities to decide the most likely or a reasonable next token.

What are N-gram Models?

N-gram models use fixed-length word sequences:

- **Bigram (2-gram):** “you are”
- **Trigram (3-gram):** “are very nice”

They predict the next word using just the previous N−1 words.
For example:
Given: **"orange is"**
Possible next words: **"ripe"** or **"cheerful"** (fruit vs. color)

Problem: They lack deep context. A 3-gram only knows 2 words before the target word, which often isn't enough to understand meaning or nuance.

Also, the more words (larger N) you use, the rarer those patterns become in the training data — leading to sparse or unhelpful predictions.

Recurrent Neural Networks (RNNs)

RNNs are a type of neural network that can handle sequences — they learn from one token at a time and carry over context from earlier tokens.

They can remember more context than N-gram models and can theoretically understand whole sentences.

Limitation: RNNs struggle with *long-range dependencies* (remembering things from far earlier in the text) due to something called the **vanishing gradient problem** — they "forget" as the sequence grows.

Why Context Matters

Just like humans understand jokes or stories by remembering earlier parts, language models perform better when they understand the *full context*.

Short-context models (like N-grams or simple RNNs) often make mistakes because they don't "remember" enough.

Final Takeaway

- **Language models predict words based on previous ones.**
- **N-grams** are simple but don't capture much context.
- **RNNs** are better, but still struggle with long text.
- **Modern large language models** (like GPT) can use entire paragraphs or documents as context, making them much more accurate.

What are Large Language Models (LLMs)

LLMs are advanced language models that can:

- Predict the next word, sentence, or paragraph.
- Understand and generate human-like text.

They work with **tokens**, which can be:

- Words
- Subwords (parts of words)

- Characters

Why Are LLMs Better Than Older Models?

1. **More parameters** (millions to trillions): They can learn deeper patterns.
2. **More context**: They can consider entire paragraphs, not just a few words.
3. **Smarter architecture**: LLMs use **Transformers**, which outperform older models like N-grams or RNNs.

What Is a Transformer?

A **Transformer** is a special kind of neural network used for understanding and generating text. It has two main parts:

- **Encoder**: Reads and understands the input.
- **Decoder**: Produces the output (e.g., a translation).

Example:
Input: *"I am a good dog."*
Output: *"Je suis un bon chien."* (French translation)

What Is Self-Attention?

Self-attention helps the model decide **which words in a sentence are most important to each other**.

Example:

“The animal didn’t cross the street because **it** was too tired.”

Which noun does **"it"** refer to — *animal* or *street*?

Self-attention helps the model focus on the correct one (in this case, **animal**).

Types of Self-Attention

- **Bidirectional**: Looks at words before **and** after a token (used in encoders).
- **Unidirectional**: Looks only at previous words (used in decoders, like in text generation).

What is Multi-Head, Multi-Layer Attention?

- Each **self-attention layer** has multiple **heads**.
- Each head learns different types of relationships (e.g., grammar, meaning, word references).
- Multiple layers help the model learn **more complex ideas**:
    - Lower layers learn grammar
    - Higher layers learn deeper meaning, emotions, or context

---

Why Are Transformers So Big?

- Larger models (with more parameters) learn more patterns and perform better.
- That's why models like GPT-4 have **hundreds of billions of parameters**.
- But: Larger size = More training time, data, energy, and cost.

How Do LLMs Generate Text?

They **predict one token at a time** based on previous ones—like a super-advanced autocomplete.

Example:

Prompt: *My dog, Max, knows how to perform many traditional dog tricks.*
LLM generates: *"For example, he can sit, stay, and roll over."*

Even when answering a user question, the model is just **predicting the next likely token** based on patterns it learned.

Pros of LLMs

Can write clear and fluent text
Handle many tasks (translation, summarization, Q&A)
Adaptable to new tasks even without direct training (to some extent)

Problems with LLMs

Hallucinations: They may give wrong or made-up answers
Bias: Can reflect unfair or harmful patterns from training data
Expensive: Training needs a LOT of data, time, energy, and powerful computers

---

What Are Foundation (Base/Pre-trained) LLMs?

A **foundation LLM** is a general-purpose language model trained on massive amounts of natural language. It "knows" a lot about:

- Grammar and structure
- Common sense and idioms
- Language patterns

It can:

- Generate fluent text
- Answer general questions
- Even write poetry

But it **cannot** directly solve:

- Regression problems (e.g., predicting a number)
- Classification tasks (e.g., labeling spam vs. non-spam)

These tasks require **fine-tuning** or **clever prompt design**.

Fine-Tuning

What it is:

Fine-tuning means retraining a **foundation LLM** on a **task-specific dataset** (e.g., sentiment classification or legal document summarization).

Why it's useful:

Even a small amount of extra training can help the LLM specialize in a new task.

How it works:

- The model's parameters (weights and biases) are **updated** during backpropagation.
- Standard fine-tuning updates **all parameters**, which can be **computationally expensive**.

Smarter Way: Parameter-Efficient Tuning

- Only **a small subset of parameters** are updated.
- Requires less compute and memory.
- Faster and more resource-friendly.

Result: Same-size model (e.g., 10B parameters), but specialized for a specific task.

Distillation

What it is:

Distillation compresses a large model (the **teacher**) into a smaller model (the **student**), which:

- Runs faster
- Uses less energy and compute
- Is easier to deploy

Tradeoff:

- Slightly lower performance than the full-size LLM.
- Good balance for real-world use cases.

Used when serving speed and cost are more important than perfect accuracy.

Prompt Engineering

Instead of training the model again, you can **instruct the model more cleverly** using examples in your prompt.

3 Types:

1. **Zero-shot prompting**:
   Give the model just a question.
   → Example: "apple: _____" (may return unrelated info)
2. **One-shot prompting**:
   Show one example.
   → "peach: drupe"
   → Then ask: "apple: _____" → Better result
3. **Few-shot prompting**:
   Show multiple examples.
   → "plum: drupe"
   → "pear: pome"
   → "lemon: _____" → Even more accurate

🩹 **Note**: Prompting **does not change** the model's parameters.

Offline Inference

What it is:

Instead of generating predictions **live (online)**, the LLM is run **ahead of time** to:

- Process data in bulk
- Cache results
- Serve results quickly when needed

Example:

Google used LLMs to generate **800+ synonyms for COVID vaccines** in over 50 languages offline. These were then used in real-time searches.

Ideal for **slow but heavy jobs** (e.g., once-a-week batch processing).

Use LLMs Responsibly

LLMs **inherit biases** from:

- The training data (often scraped from the internet)
- The fine-tuning or distillation datasets

Responsible Use:

- Always consider **fairness and ethical impact**
- Follow established **fairness guidelines**
- Avoid reinforcing harmful or unjust societal biases

Responsible Use Watch for and mitigate bias, be transparent, use fair datasets

---

2. Project Work:

2.1 Breast Cancer Prediction

Objective

Predict whether a tumor is malignant or benign based on diagnostic features.

Methodology

1. **Data Preprocessing**
    - Dropped fully null columns (e.g., 'Unnamed')
    - Handled missing values with suitable imputation strategies
    - Applied log transformation to skewed features
    - Standardized numerical features using StandardScaler
2. **Model Training**
    - Trained Logistic Regression, Random Forest, SVC, and KNN models
    - Used pipelines for consistent preprocessing and training
    - Skipped PCA to retain interpretability for feature importance
3. **Evaluation**
    - Used **accuracy**, **confusion matrix**, **recall**, and **ROC-AUC** for performance assessment
    - Random Forest performed best overall with high accuracy and interpretability
4. **Feature Importance**
    - Identified top features: texture_worst, radius_se, and symmetry_worst
    - These features showed the strongest contribution in distinguishing between benign and malignant tumors

---

2.2 Fashion Mnist Dataset:

Objective

Classify clothing items (e.g., shirts, shoes, bags) into one of 10 fashion categories based on image pixel data.

Methodology

1. **Data Preprocessing**
    - Loaded grayscale images from the Fashion MNIST dataset
    - Normalized pixel values to the [0, 1] range
    - Reshaped data to match CNN input shape (28×28×1)
    - One-hot encoded the target labels for multi-class classification
2. **Model Training**
    - Built multiple models:
      • Basic CNN
      • CNN with Dropouts
    - Used Adam optimizer and categorical_crossentropy loss
    - Trained models using early stopping and validation split for generalization
3. **Evaluation**
    - Evaluated performance using accuracy, confusion matrix, and classification report
    - Used 5-fold cross-validation on a 10,000-sample subset for reliability
    - CNN with Dropout showed strong accuracy with reduced overfitting

3. Challenges and Approaches:

3.1 Breast Cancer Prediction

- Overall the dataset was clean and no major preprocessing was required.
- I dropped the unnamed: 32 column because it was completely empty and filled with null values.
- As the data was sort of linear so I trained Logistic Regression along with SVC, KNeighborsClassifier, Random Forest etc.

3.2 Fashion Mnist Dataset

- The dataset was well-structured and did not contain missing or null values, which reduced the need for heavy preprocessing.
- Since the data consisted of grayscale images (28x28 pixels), normalization was applied to scale pixel values between 0 and 1, improving convergence during training.
- Traditional machine learning models struggled with performance due to the complex patterns in image data.
- A Convolutional Neural Network (CNN) was implemented to better capture spatial hierarchies and texture features in the images.
- Dropout layers were added at multiple levels to reduce overfitting and enhance generalization on unseen data.
- ReLU was used as the activation function in hidden layers, while softmax was used in the final layer for multi-class classification.

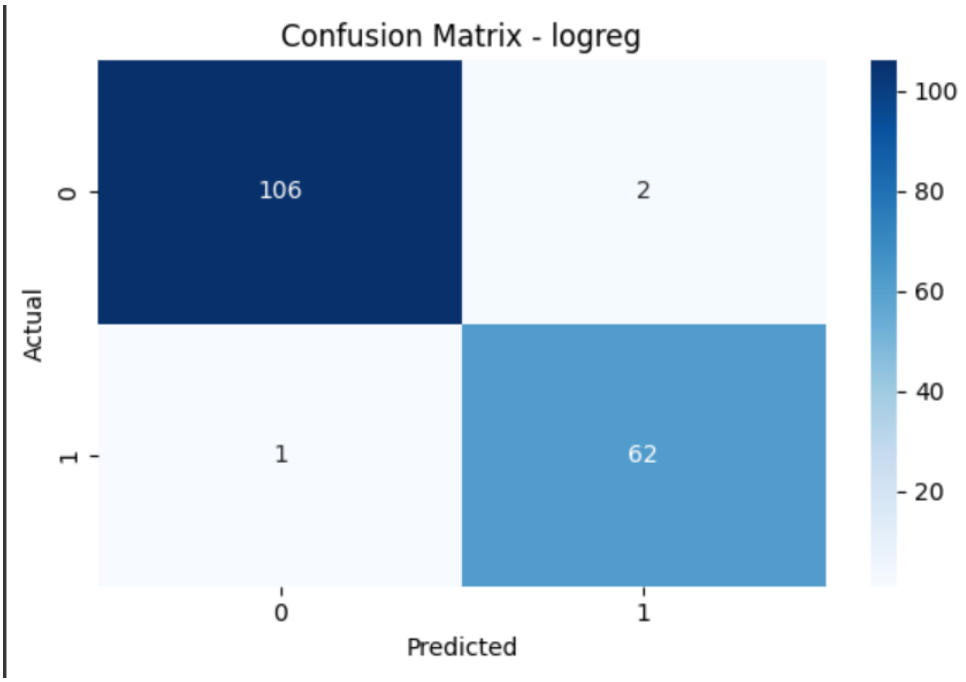
4. Results and Analysis:

4.1 Breast Cancer Prediction

Classification Report:

	precision	recall	f1-score	support
0	0.99	0.98	0.99	108
1	0.97	0.98	0.98	63
accuracy			0.98	171
macro avg	0.98	0.98	0.98	171
weighted avg	0.98	0.98	0.98	171

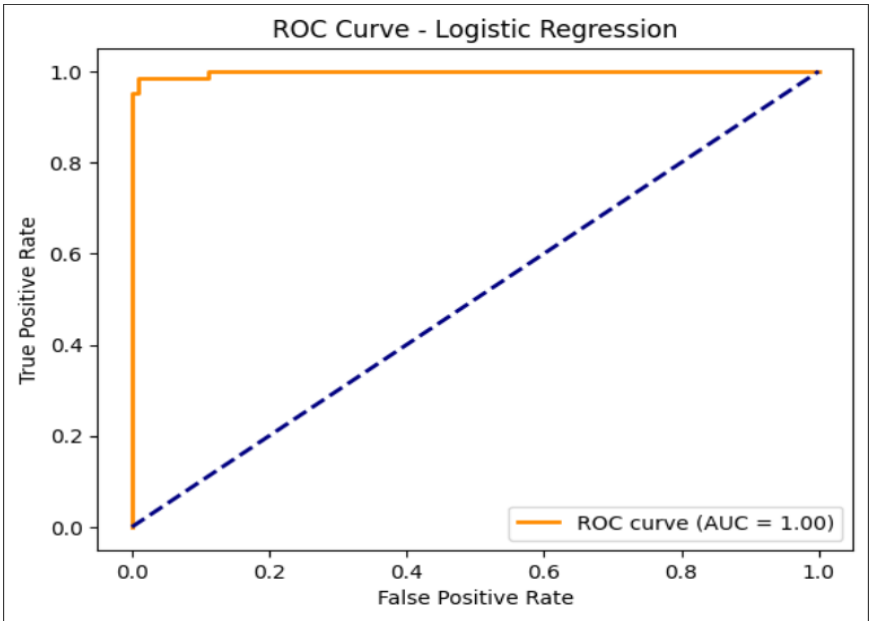
Confusion Matrix:



Accuracy: 0.9824561403508771

AUC-SCORE: 0.9828042328042329

AUC_ROC Curve:



4.2 Fashion Mnist dataset

Accuracy: 0.92

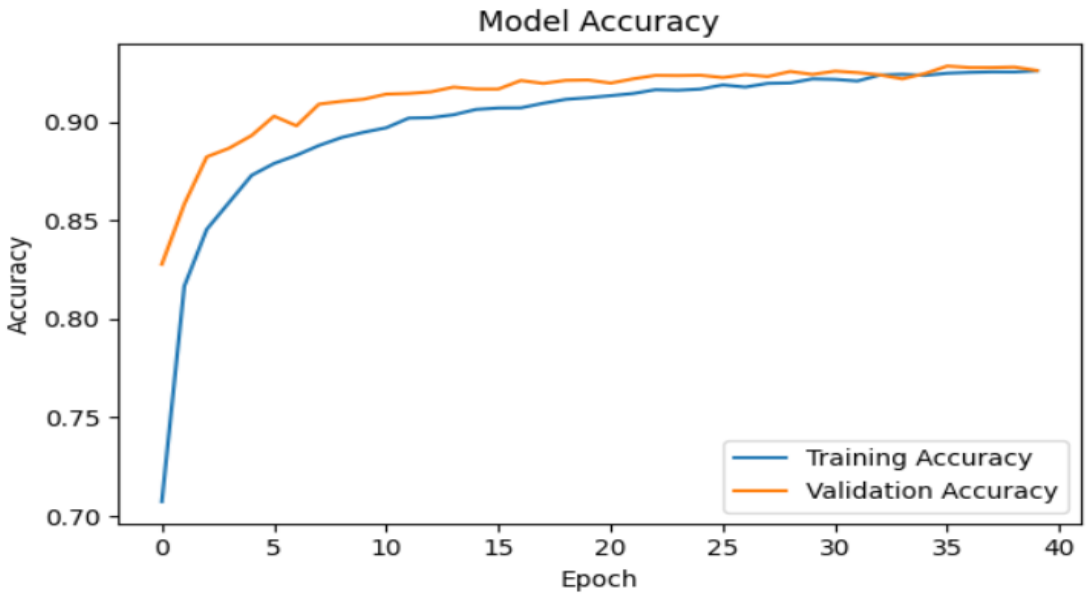
Test loss: 0.23

Test accuracy: 0.91

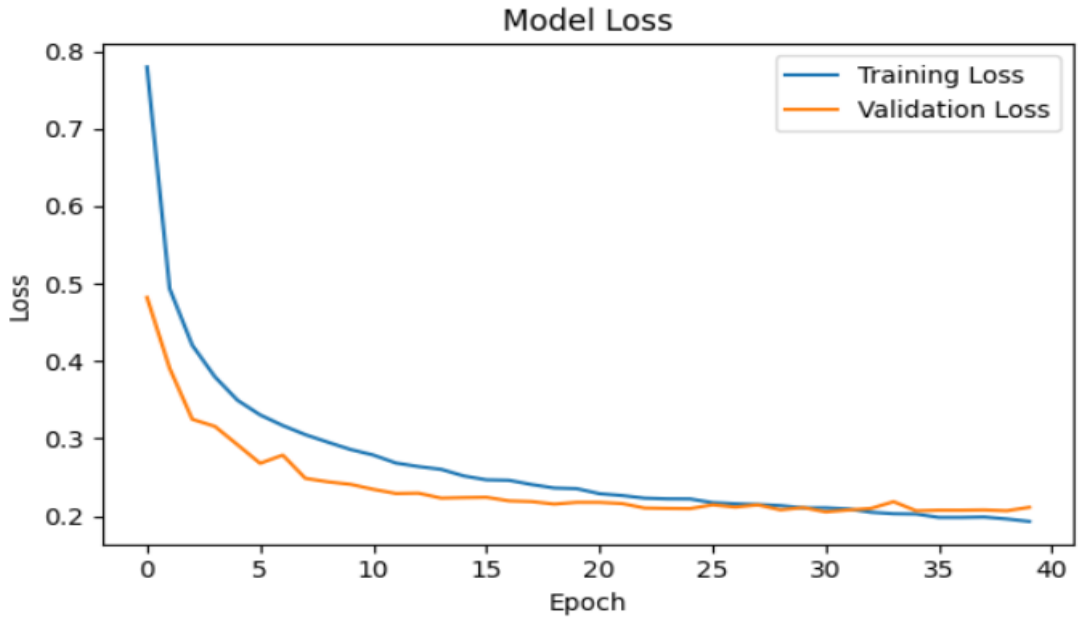
Classification Report:

	precision	recall	f1-score	support
T-shirt/top	0.90	0.81	0.85	1000
Trouser	0.99	0.98	0.99	1000
Pullover	0.90	0.86	0.88	1000
Dress	0.91	0.92	0.91	1000
Coat	0.85	0.89	0.87	1000
Sandal	0.99	0.99	0.99	1000
Shirt	0.72	0.78	0.75	1000
Sneaker	0.96	0.97	0.97	1000
Bag	0.99	0.98	0.98	1000
Ankle boot	0.98	0.97	0.97	1000
accuracy			0.91	10000
macro avg	0.92	0.91	0.92	10000
weighted avg	0.92	0.91	0.92	10000

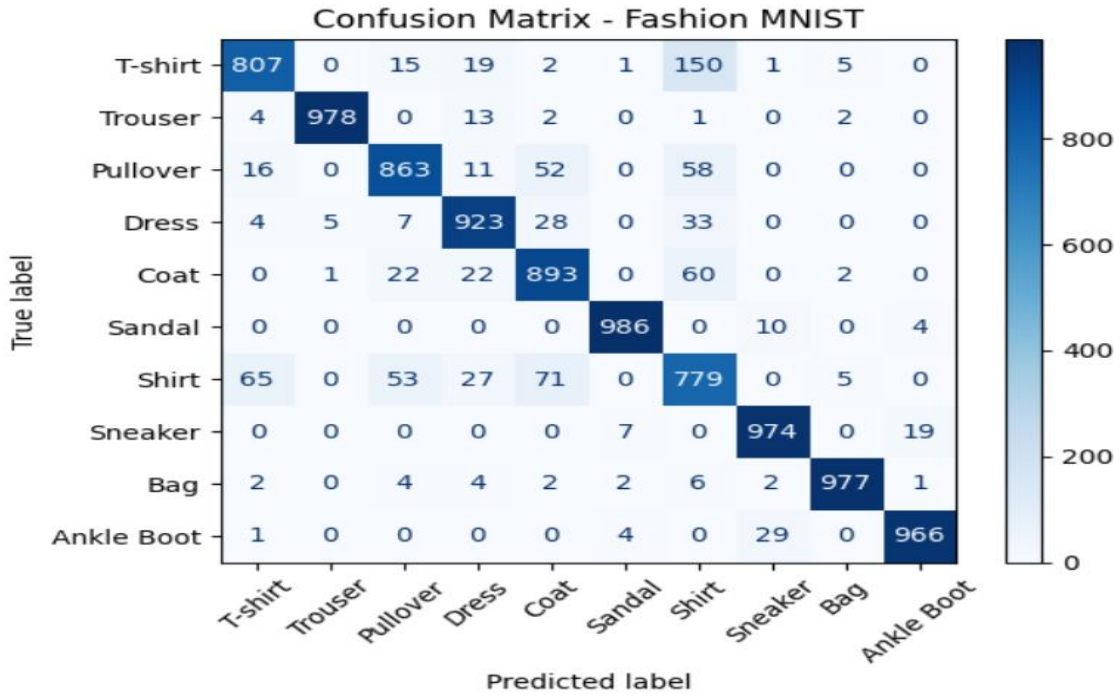
Model Accuracy Curve:



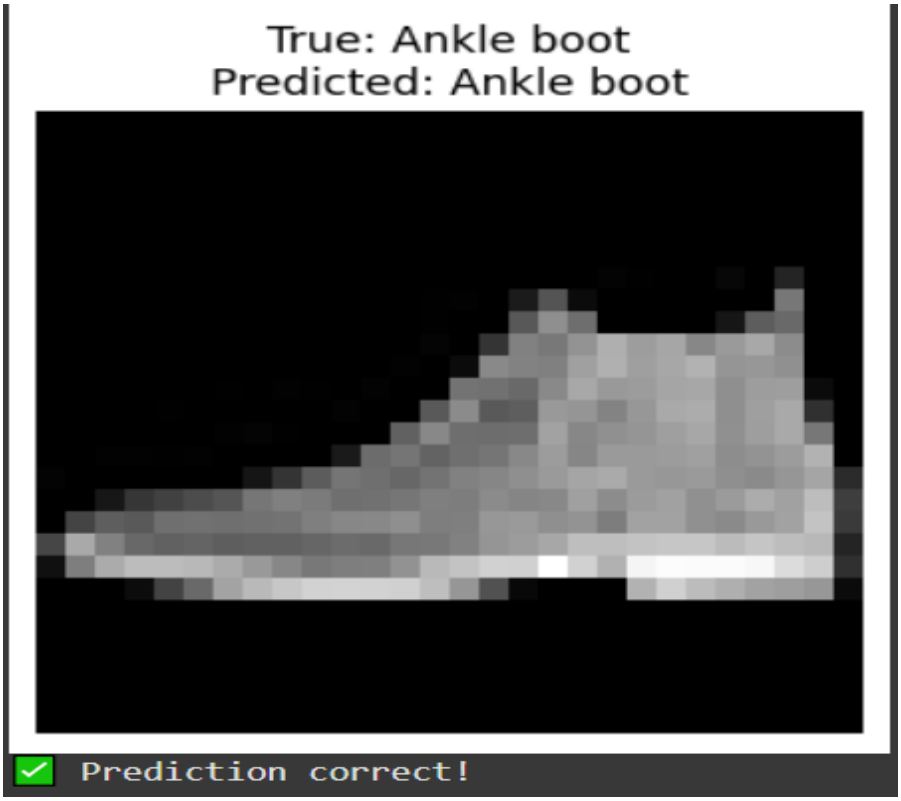
Training Loss vs Validation Loss Curve:



Confusion Matrix:



Prediction for 1st Image:



5. Conclusion:

Task: 1 Breast Cancer Prediction

The model demonstrates promising performance in predicting breast cancer using key diagnostic features. However, the dataset used is relatively small, which may limit the model's generalizability in real-world scenarios. In the future, collecting a larger and more diverse dataset will be essential to improve the model's robustness and reliability. Training on more data will help reduce overfitting and enhance the model's ability to generalize to unseen cases in clinical settings.

Task: 2 Fashion MNIST Classification

The model shows strong performance in classifying images from the Fashion MNIST dataset using various machine learning algorithms. The dataset is well-structured and balanced, allowing effective training and evaluation. However, while the model performs well on this benchmark dataset, real-world fashion images often have more complexity, variety, and noise. To improve robustness and real-world applicability, future work should involve training on larger, more diverse, and high-resolution fashion datasets. This would enhance the model's ability to generalize to more complex clothing recognition tasks.

6. References:

- https://scikit-learn.org/stable/
- https://developers.google.com/machine-learning/crash-course/
- https://www.youtube.com/
- https://www.youtube.com/@campusx-official
- https://keras.io/api/layers/convolution_layers/
- https://www.kaggle.com/code/gpreda/cnn-with-tensorflow-keras-for-fashion-mnist/