ISDM (INDEPENDENT SKILL DEVELOPMENT MISSION

◇ UNDERSTANDING NEURAL NETWORKS: PERCEPTRON, ACTIVATION FUNCTIONS

📌 INTRODUCTION

A **Neural Network** is a computational model inspired by the structure of the human brain. It consists of interconnected **neurons (nodes)** that process and learn from data. Neural networks form the foundation of **Deep Learning**, enabling tasks like **image recognition, speech processing, and natural language understanding**.

Why Neural Networks?

✓ **Learn from Data** – No need for manual feature extraction.
✓ **Handle Complex Problems** – Used in **computer vision, robotics, and AI applications**.
✓ **Adaptability** – Improves performance as more data is provided.

In this study material, we will explore the **Perceptron model**, various **Activation Functions**, and how they contribute to the functioning of Neural Networks.

📌 CHAPTER 1: UNDERSTANDING THE PERCEPTRON MODEL

1.1 What is a Perceptron?

A **Perceptron** is the **simplest type of artificial neuron** used in Machine Learning. It is a **linear classifier** that maps input data to an output using a weighted sum and an activation function.

✓ It is the **building block of neural networks**.
✓ It is used for **binary classification problems** (e.g., Spam vs. Not Spam).

- **Example:**

A **perceptron-based email classifier** can determine if an email is **spam (1) or not spam (0)** based on keywords.

1.2 Structure of a Perceptron

A perceptron consists of:

1. **Inputs (X)** – The features of the dataset.
2. **Weights (W)** – Adjusted values to influence input importance.
3. **Summation Function** – Computes weighted sum:

$$Z = (W_1X_1 + W_2X_2 + ... + W_nX_n) + Bias$$

4. **Activation Function (f)** – Determines the output:

$$Y = f(Z)$$

1.3 Types of Perceptrons

- **Single-layer Perceptron** – Used for simple binary classification tasks.

- **Multi-layer Perceptron (MLP)** – Contains multiple perceptrons, used for complex problems.

Example: A **single-layer perceptron** can classify if a fruit is an **apple or orange**, whereas a **multi-layer perceptron** can classify **multiple fruit types**.

1.4 Perceptron Learning Algorithm

The **Perceptron algorithm** follows these steps:

1. **Initialize weights and bias** randomly.
2. **Compute weighted sum** of inputs.
3. **Apply activation function** to determine output.
4. **Adjust weights** using error correction (if prediction is wrong).
5. **Repeat until convergence** (small error rate).

1.5 Limitations of a Perceptron

❌ **Cannot solve non-linear problems** (e.g., XOR problem).
❌ **Limited to simple classification tasks**.
❌ **Sensitive to outliers** in data.

To overcome these issues, advanced models like **Multi-layer Perceptrons (MLPs) and Deep Neural Networks (DNNs)** are used.

📌 Chapter 2: Understanding Activation Functions

2.1 What is an Activation Function?

An **Activation Function** determines whether a neuron should be activated or not based on input values. It **adds non-linearity** to neural networks, enabling them to learn complex patterns.

✓ Without activation functions, neural networks behave like **linear models**.
✓ Different activation functions are used based on **task complexity**.

2.2 Types of Activation Functions

Activation Function	Formula	Use Case
Step Function	$f(x) = 1$ if $x \geq 0$, else 0	Used in Perceptrons for binary classification
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	Probability-based outputs (e.g., Logistic Regression)
Tanh (Hyperbolic Tangent)	$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	Image Processing, NLP tasks
ReLU (Rectified Linear Unit)	$f(x) = \max(0, x)$	Deep learning, CNNs
Leaky ReLU	$f(x) = x$ if $x > 0$, else $0.01x$	Handles negative values better than ReLU
Softmax	$f(x_i) = \frac{e^{x_i}}{\sum e^{x_j}}$	Multi-class classification

---

2.3 Properties & Uses of Activation Functions

✓ Step Function:

- Good for binary classification.
- Not useful in deep learning (produces discrete outputs).

✓ Sigmoid Function:

- Outputs values between **0 and 1**.
- Used in **logistic regression and neural networks**.

- **Disadvantage:** Causes **vanishing gradient problem,** making training slow.

✓ Tanh Function:

- Similar to Sigmoid but ranges from **-1 to 1**.
- Better for **zero-centered data,** commonly used in **RNNs**.

✓ ReLU Function:

- Outputs **0 for negative inputs** and **x for positive inputs**.
- Used in **deep learning models** due to its efficiency.
- **Disadvantage:** Cannot handle negative values (dying ReLU problem).

✓ Softmax Function:

- Converts outputs into **probabilities for multi-class classification**.
- Commonly used in the **final layer** of neural networks.

2.4 Choosing the Right Activation Function

Scenario	Best Activation Function
Binary Classification	Sigmoid, Step Function
Multi-Class Classification	Softmax
Deep Learning (CNNs, RNNs)	ReLU, Leaky ReLU
Regression Problems	Linear Activation

📌 Chapter 3: Implementing a Simple Neural Network with Perceptron & Activation Functions

Let's implement a **single-layer perceptron** using Python with **NumPy**.

3.1 Import Libraries

```
import numpy as np
```

3.2 Define Perceptron Model

```
def perceptron(x, w, b):
    weighted_sum = np.dot(x, w) + b
    return np.where(weighted_sum >= 0, 1, 0)  # Step Activation
```

3.3 Initialize Inputs & Weights

```
X = np.array([[0,0], [0,1], [1,0], [1,1]])  # Input features
W = np.array([0.5, 0.5])  # Weights
B = -0.7  # Bias
```

3.4 Compute Output

```
output = perceptron(X, W, B)
print(output)
```

✓ This simple model **predicts binary outputs** based on input conditions.

📌 Chapter 4: Exercises & Assignments

4.1 Multiple Choice Questions (MCQs)

1. **Which activation function is commonly used in Deep Learning?**
    - (a) Step Function
    - (b) Sigmoid
    - (c) ReLU ✅
    - (d) Softmax
2. **What is the main limitation of the Perceptron model?**
    - (a) It works only for linear problems ✅
    - (b) It requires large datasets
    - (c) It is too fast to train
    - (d) It cannot process numerical inputs

4.2 Practical Assignments

📌 **Task 1:** Implement a **Sigmoid Activation Function** in Python and test it on sample values.

📌 **Task 2:** Modify the **Perceptron Model** to use **ReLU activation** instead of the Step function.

📌 **Task 3:** Build a **Multi-layer Perceptron (MLP) model** using TensorFlow/Keras.

📜 SUMMARY

✓ **Perceptrons** are the building blocks of neural networks, used for **binary classification**.

✓ **Activation Functions** introduce non-linearity, making neural networks **more powerful**.

✓ **ReLU and Softmax** are widely used in **Deep Learning models**.

✓ **Understanding perceptrons and activation functions** is essential for learning **advanced neural networks and deep learning techniques**.

◇ Introduction to Deep Learning & TensorFlow/Keras

📌 Introduction

Deep learning is a subset of machine learning that focuses on artificial neural networks designed to **mimic the human brain**. It enables computers to **recognize patterns, make decisions, and learn from data** without explicit programming. Deep learning is widely used in **computer vision, natural language processing (NLP), healthcare, robotics, and many other fields**.

In this study material, we will cover: ✓ The **fundamentals of deep learning**
✓ **Neural networks** and how they work
✓ **Introduction to TensorFlow and Keras**, two of the most widely used deep learning frameworks

---

📌 Chapter 1: Understanding Deep Learning

1.1 What is Deep Learning?

Deep learning is a branch of **artificial intelligence (AI)** that enables computers to learn from large amounts of data. It is based on **multi-layered neural networks**, which can **automatically extract patterns and features** from raw data.

🔹 **Key Features of Deep Learning:** ✓ Works well with large datasets
✓ Uses multiple layers of artificial neurons
✓ Capable of **self-learning and feature extraction**
✓ Requires **powerful hardware (GPUs, TPUs)** for training

🔹 **Applications of Deep Learning:** ✓ **Computer Vision:** Face recognition, medical imaging, self-driving cars

✓ **Natural Language Processing (NLP):** Chatbots, speech recognition, language translation

✓ **Healthcare:** Disease detection, drug discovery, AI-assisted diagnosis

✓ **Finance:** Fraud detection, stock market prediction

1.2 Deep Learning vs. Traditional Machine Learning

Feature	Machine Learning	Deep Learning
Feature Engineering	Requires manual feature extraction	Learns features automatically
Performance with Big Data	Limited effectiveness	Improves with more data
Hardware Requirement	Runs on CPUs	Requires GPUs/TPUs
Interpretability	More interpretable	Black-box nature
Use Cases	Predictive analytics, structured data	Image recognition, speech processing

---

📌 Chapter 2: Understanding Neural Networks

2.1 What are Artificial Neural Networks (ANNs)?

Artificial Neural Networks (ANNs) are inspired by the structure and function of the human brain. They consist of **artificial neurons** that are organized into layers.

- **Neural Network Structure: ✓ Input Layer** – Receives raw data

✓ **Hidden Layers** – Processes and extracts patterns

✓ **Output Layer** – Provides the final decision/prediction

- **How Neurons Work:**

1. Each neuron receives inputs (features).
2. Weights are applied to the inputs.
3. A mathematical function (activation function) determines whether the neuron "fires".
4. The processed information is passed to the next layer.

- **Types of Neural Networks: ✓ Feedforward Neural Networks (FNNs):** Simple networks for classification and regression.

✓ **Convolutional Neural Networks (CNNs):** Used for image processing.

✓ **Recurrent Neural Networks (RNNs):** Works with sequential data (e.g., speech, text).

2.2 The Training Process: Forward & Backpropagation

- **Forward Propagation:**

✓ Inputs pass through layers to generate predictions.

- **Backpropagation:**

✓ The model adjusts weights using **gradient descent** to minimize error.

✓ The process repeats until the error is minimized.

- **Loss Function & Optimization:**

✓ **Loss function** measures prediction error (e.g., Mean Squared

Error, Cross-Entropy).
✓ **Optimizers** (e.g., Adam, SGD) adjust weights to reduce loss.

📌 Chapter 3: Introduction to TensorFlow & Keras

3.1 What is TensorFlow?

TensorFlow is an **open-source deep learning framework** developed by Google. It allows developers to build and train neural networks efficiently.

- **Key Features:** ✓ Supports **GPU acceleration** for faster computations
✓ Works with **Numpy, Pandas, and other Python libraries**
✓ Provides **automatic differentiation** for optimizing neural networks

- **Common TensorFlow Applications:** ✓ Image recognition
✓ Speech-to-text processing
✓ Time-series forecasting
✓ Text generation

3.2 What is Keras?

Keras is a **high-level deep learning API** built on top of TensorFlow. It provides an easy-to-use interface for **building, training, and testing deep learning models**.

- **Why Use Keras? ✓ User-friendly and modular**
✓ Supports rapid prototyping
✓ Integrates seamlessly with TensorFlow

📌 Chapter 4: Building a Simple Neural Network with TensorFlow/Keras

4.1 Installing TensorFlow & Keras

Before using TensorFlow and Keras, install them using:

pip install tensorflow

Import necessary libraries:

import tensorflow as tf

from tensorflow import keras

import numpy as np

4.2 Creating a Simple Neural Network

Let's create a simple neural network using **Keras Sequential API**.

```
# Import TensorFlow & Keras

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense


# Define a simple neural network

model = Sequential([
```

```
  Dense(16, activation='relu', input_shape=(10,)),  # Input layer with 10 features
  Dense(8, activation='relu'),  # Hidden layer
  Dense(1, activation='sigmoid')  # Output layer (Binary classification)
])

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

# Display model summary
model.summary()
```

✓ **Explanation:**

- The model has **three layers** (input, hidden, and output).
- **ReLU activation** is used for hidden layers.
- **Sigmoid activation** is used for binary classification.
- The **Adam optimizer** helps adjust weights efficiently.

4.3 Training the Model

```
# Generate dummy training data
X_train = np.random.rand(1000, 10)  # 1000 samples, 10 features
```

```
y_train = np.random.randint(0, 2, 1000)  # 1000 binary labels (0 or 1)


# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32)
```

✓ **Explanation:**

- **1000 samples** with **10 features** are randomly generated.
- The model **trains for 10 epochs** using batch size **32**.
- Loss and accuracy are updated at each epoch.

4.4 Evaluating the Model

```
# Generate test data
X_test = np.random.rand(200, 10)
y_test = np.random.randint(0, 2, 200)


# Evaluate model performance
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy:.2f}")
```

✓ **Explanation:**

- The model is tested on **200 new samples**.
- **Accuracy score** indicates how well the model generalizes to new data.

📌 Chapter 5: Exercises & Assignments

5.1 Multiple Choice Questions (MCQs)

1. **Which deep learning framework is developed by Google?**
   - (a) PyTorch
   - (b) Scikit-learn
   - (c) TensorFlow ✅
   - (d) NumPy
2. **Which activation function is commonly used in output layers for binary classification?**
   - (a) ReLU
   - (b) Sigmoid ✅
   - (c) Softmax
   - (d) Tanh
3. **What is the primary advantage of using GPUs in deep learning?**
   - (a) Faster computations ✅
   - (b) Lower power consumption
   - (c) Increased storage capacity
   - (d) Better visualization

5.2 Practical Assignments

📌 **Task 1:** Build a **deep learning model** using Keras for image classification.

📌 **Task 2:** Train a **simple neural network** on the MNIST dataset.

📌 **Task 3:** Experiment with different **activation functions (ReLU, Sigmoid, Softmax)**.

📜 SUMMARY

✓ **Deep Learning** enables machines to learn complex patterns from data.

✓ **Neural Networks** consist of **input, hidden, and output layers**.

✓ **TensorFlow & Keras** are widely used for deep learning model development.

✓ **PCA and t-SNE** help visualize high-dimensional data in 2D.

◇ Building Feedforward Neural Networks

📌 Introduction

A **Feedforward Neural Network (FNN)** is the simplest type of artificial neural network where the flow of information moves **only in one direction**—from input to output, without any loops or cycles. These networks are widely used for classification, regression, and pattern recognition tasks.

Why are Feedforward Neural Networks Important?

✓ **Used in Various Applications** – Image recognition, speech processing, medical diagnostics.
✓ **Foundation of Deep Learning** – Forms the basis of advanced deep neural networks.
✓ **Easy to Implement** – Simple architecture with clear connections between layers.
✓ **Universal Approximation** – Can model any function given enough neurons and layers.

In this study material, we will **explore the architecture, working principles, training process, and real-world applications** of Feedforward Neural Networks.

📌 Chapter 1: Understanding Feedforward Neural Networks

1.1 What is a Feedforward Neural Network?

A **Feedforward Neural Network (FNN)** is a neural network where **information flows from input to output in a single direction** without cycles. The primary goal of an FNN is to **map input data to output predictions** using layers of neurons.

1.2 Basic Structure of a Feedforward Neural Network

A typical **FNN consists of three main layers:**

1. **Input Layer** – Receives input data (e.g., images, text, numerical data).
2. **Hidden Layers** – Perform computations, transforming inputs into meaningful features.
3. **Output Layer** – Produces final predictions (e.g., classification labels, numerical outputs).

1.3 How Feedforward Neural Networks Work

- Each neuron in a layer receives inputs, applies weights, and passes the result through an **activation function**.
- The output from one layer **becomes the input for the next layer**.
- The final layer generates predictions based on learned patterns in the data.

📌 Chapter 2: Activation Functions in Feedforward Networks

2.1 What is an Activation Function?

An **activation function** introduces non-linearity into a neural network, allowing it to **learn complex patterns**.

2.2 Common Activation Functions Used in FNNs

1. **Sigmoid Activation** – Outputs values between 0 and 1, useful for binary classification.
    - Formula: f(x)=11+e−xf(x) = \frac{1}{1 + e^{-x}}
    - Issue: Prone to **vanishing gradient problem** in deep networks.
2. **ReLU (Rectified Linear Unit)** – Most commonly used; outputs **0** for negative values and **x** for positive values.
    - Formula: f(x)=max⁡(0,x)f(x) = \max(0, x)
    - Benefits: Overcomes vanishing gradients, **faster training**.
3. **Softmax Activation** – Used in multi-class classification to **normalize output probabilities**.
4. **Tanh Activation** – Outputs between -1 and 1, making it useful for centered data.

📌 Chapter 3: Training a Feedforward Neural Network

3.1 Forward Propagation

In forward propagation, input data **flows through the network** layer by layer, with weights and activation functions transforming the data into predictions.

3.2 Loss Function

The **loss function** measures how far the predictions deviate from the actual values. Common loss functions:

- **Mean Squared Error (MSE)** – Used for regression tasks.
- **Cross-Entropy Loss** – Used for classification problems.

3.3 Backpropagation Algorithm

To improve predictions, neural networks use **backpropagation**, which works as follows:

1. Compute the loss using the loss function.
2. Calculate the gradients of weights using the **chain rule of differentiation**.
3. Update weights using an **optimizer (e.g., Gradient Descent, Adam Optimizer)**.

3.4 Optimizers for Feedforward Networks

- **Gradient Descent** – Updates weights in the direction that minimizes error.
- **Adam (Adaptive Moment Estimation)** – Adaptive learning rate for faster convergence.

📌 Chapter 4: Implementing a Feedforward Neural Network in Python

4.1 Steps to Build a Feedforward Neural Network

1. **Load Dataset** – Choose a dataset (e.g., MNIST, Iris dataset).
2. **Preprocess Data** – Normalize, split into training and testing sets.
3. **Define the Neural Network** – Specify layers, neurons, and activation functions.

4. **Train the Model** – Use forward propagation, backpropagation, and optimization.
5. **Evaluate the Model** – Test on unseen data to check performance.

4.2 Python Code Implementation

Here's a basic Feedforward Neural Network using **TensorFlow and Keras**:

```
# Install TensorFlow if not installed
!pip install tensorflow


# Import required libraries
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import numpy as np


# Load sample dataset (Iris dataset)
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
# Load and preprocess data
iris = load_iris()
X = iris.data
y = keras.utils.to_categorical(iris.target, num_classes=3)


# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


# Define a simple Feedforward Neural Network
model = Sequential([
  Dense(10, activation='relu', input_shape=(4,)),  # Hidden layer with ReLU
  Dense(8, activation='relu'),  # Another hidden layer
  Dense(3, activation='softmax')  # Output layer for 3 classes
])
```

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


# Train the model
model.fit(X_train, y_train, epochs=50, batch_size=5, validation_data=(X_test, y_test))


# Evaluate model
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy:.2f}")
```

✓ **Output:** Trains a Feedforward Neural Network to classify flowers with **high accuracy**.

📌 Chapter 5: Applications of Feedforward Neural Networks

5.1 Real-World Use Cases

1. **Handwritten Digit Recognition** – Used in postal services (e.g., reading zip codes).
2. **Medical Diagnosis** – Detecting diseases from patient data.
3. **Stock Price Prediction** – Analyzing trends in stock markets.
4. **Spam Email Detection** – Classifying emails as spam or legitimate.

5. **Speech Recognition** – Converting voice into text commands.

📌 Chapter 6: Exercises & Assignments

6.1 Multiple Choice Questions (MCQs)

1. **Which activation function is commonly used in hidden layers of FNNs?**
   - (a) Sigmoid
   - (b) ReLU ✅
   - (c) Softmax
   - (d) Linear
2. **What is the role of backpropagation in neural networks?**
   - (a) Forward data flow
   - (b) Compute loss
   - (c) Update weights using gradients ✅
   - (d) Remove unnecessary neurons
3. **Which optimizer is commonly used for training deep neural networks?**
   - (a) Gradient Descent
   - (b) Adam Optimizer ✅
   - (c) SGD
   - (d) Newton's Method

6.2 Practical Assignments

- **Task 1:** Modify the Python implementation to classify handwritten digits (use the MNIST dataset).
- **Task 2:** Train an FNN to classify **customer churn data**.
- **Task 3:** Research how **feedforward networks** are used in **autonomous driving**.

📜 Summary & Key Takeaways

- **Feedforward Neural Networks** are the simplest type of artificial neural networks.
- **Activation functions** introduce non-linearity and help in learning complex patterns.
- **Backpropagation and optimization algorithms** improve the network's accuracy.
- **FNNs are widely used** in **image classification, fraud detection, speech recognition, and medical diagnosis**.

◇ Convolutional Neural Networks (CNNs) for Image Recognition

📌 Introduction

In the field of **Deep Learning, Convolutional Neural Networks (CNNs)** have revolutionized **image recognition and computer vision** tasks. Unlike traditional machine learning models, CNNs are designed to automatically detect **patterns, edges, textures, and shapes** in images without the need for manual feature extraction.

CNNs are widely used in:

✓ **Face Recognition** (e.g., Face ID on smartphones)

✓ **Medical Imaging** (e.g., Detecting tumors in X-rays)

✓ **Self-Driving Cars** (e.g., Object detection for road safety)

✓ **Security & Surveillance** (e.g., Intruder detection in CCTV footage)

In this study material, we will explore the **structure, working, and applications of CNNs**, along with practical implementation examples.

📌 Chapter 1: Understanding Convolutional Neural Networks (CNNs)

1.1 What is a CNN?

A **Convolutional Neural Network (CNN)** is a **deep learning architecture** designed to process structured grid-like data such as images. Unlike traditional **Artificial Neural Networks (ANNs)**, CNNs use specialized layers to detect patterns and features within an image.

✓ **Traditional ANN**: Treats images as a **1D array,** leading to high computational costs.
✓ **CNN**: Processes images as **2D or 3D structures,** preserving spatial information.

🔹 **Example**:
A **CNN-based face recognition model** detects features like **eyes, nose, and mouth,** learning how they are positioned relative to each other.

1.2 Key Advantages of CNNs

✅ **Automatic Feature Extraction** – No need for manual feature selection.
✅ **Reduces Computational Complexity** – Uses filters instead of processing entire images at once.
✅ **Translation Invariance** – Can recognize objects regardless of their position in an image.
✅ **High Accuracy in Image Tasks** – Outperforms traditional ML models in vision-related tasks.

📌 Chapter 2: Architecture of a CNN

A CNN consists of multiple **layers,** each performing a specific task to **analyze, detect, and classify** images. The main components of a CNN are:

2.1 Convolutional Layer

✓ This layer **extracts features** like edges, shapes, and patterns.
✓ Uses **filters (kernels)** to detect patterns in small image regions.
✓ **Mathematical operation:**

$$\text{Feature Map} = \text{Input Image} * \text{Filter}$$

✓ **Example**: Detecting **edges in an image**.

2.2 Activation Function (ReLU - Rectified Linear Unit)

✓ Introduces **non-linearity** into the model.
✓ Converts **negative values to zero,** keeping only useful positive values.
✓ **Formula:**

$$f(x) = \max(0, x)$$

✓ **Why ReLU?**

- Helps CNN **learn complex patterns**.
- Reduces computation time compared to traditional activation functions (e.g., sigmoid).

2.3 Pooling Layer

✓ **Reduces the size of feature maps,** making computation more efficient.
✓ **Types of pooling:**

- **Max Pooling**: Selects the maximum value in a region.
- **Average Pooling**: Computes the average value in a region.

✓ **Example**: If we apply **Max Pooling** on a 2×2 region:

$$\begin{bmatrix} 1 & 3 \\ 2 & 6 \end{bmatrix}$$

The **maximum value (6) is retained,** reducing the image size while preserving important information.

2.4 Fully Connected Layer (FC Layer)

✓ Converts extracted features into a **final classification output**.
✓ **Example**: Classifies an image as **"Cat" or "Dog"** based on features extracted by convolutional layers.

📌 Chapter 3: How CNNs Work in Image Recognition?

CNNs follow **four main steps** to recognize images:

3.1 Step 1: Image Preprocessing

✓ Resize images to a standard size (e.g., **224x224 pixels** for ImageNet models).
✓ Normalize pixel values between **0 and 1** to improve performance.
✓ Convert RGB images into **grayscale or three-channel format** as required.

3.2 Step 2: Feature Extraction using Convolutional Layers

✓ Detects patterns like **edges, corners, and textures** in the first layers.

✓ Deeper layers recognize **complex patterns** like shapes and objects.

✓ **Example**:

- First layer detects **edges**.
- Second layer detects **shapes** (e.g., eyes, ears).
- Final layer recognizes **objects** (e.g., "This is a cat").

3.3 Step 3: Classification using Fully Connected Layers

✓ Uses the extracted features to classify objects.
✓ Outputs a **probability score for each category** (e.g., **90% Cat, 10% Dog**).

3.4 Step 4: Improving Accuracy with Training

✓ CNN models **learn from labeled images** using large datasets like **ImageNet, MNIST, CIFAR-10**.
✓ Model parameters are updated using **backpropagation and gradient descent**.

📌 Chapter 4: Implementing a CNN for Image Recognition in Python

We will build a **CNN model using TensorFlow and Keras** to classify images from the **MNIST dataset (handwritten digits 0-9)**.

4.1 Step 1: Install Required Libraries

```
!pip install tensorflow keras numpy matplotlib
```

4.2 Step 2: Import Libraries & Load Data

```
import tensorflow as tf

from tensorflow import keras

import numpy as np

import matplotlib.pyplot as plt


# Load the MNIST dataset

mnist = keras.datasets.mnist

(X_train, y_train), (X_test, y_test) = mnist.load_data()


# Normalize pixel values between 0 and 1

X_train, X_test = X_train / 255.0, X_test / 255.0
```

4.3 Step 3: Build the CNN Model

```
model = keras.Sequential([

  keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(28, 28, 1)),

  keras.layers.MaxPooling2D(2,2),

  keras.layers.Conv2D(64, (3,3), activation='relu'),
```

```
    keras.layers.MaxPooling2D(2,2),
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

4.4 Step 4: Train and Evaluate the Model

```
# Train the model
model.fit(X_train, y_train, epochs=5, validation_data=(X_test,
y_test))

# Evaluate on test data
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_acc:.2f}")
```

📌 Chapter 5: Real-World Applications of CNNs

✓ **Self-Driving Cars** – Detecting pedestrians, traffic lights, and obstacles.

✓ **Medical Imaging** – Identifying cancer cells in X-rays and MRIs.
✓ **Security & Surveillance** – Facial recognition for identity verification.
✓ **E-commerce** – Image-based product recommendations (e.g., Amazon, Pinterest).

📌 Chapter 6: Exercises & Assignments

6.1 Multiple Choice Questions (MCQs)

1. **What is the purpose of a Convolutional Layer?**
   - (a) Detect features like edges and shapes ✅
   - (b) Store weights for backpropagation
   - (c) Reduce the dataset size
   - (d) Apply transformations to numerical values
2. **Which activation function is commonly used in CNNs?**
   - (a) Sigmoid
   - (b) Tanh
   - (c) ReLU ✅
   - (d) Linear

6.2 Practical Assignments

📌 **Task 1:** Train a CNN on the **CIFAR-10 dataset** to classify objects (cars, animals, airplanes).
📌 **Task 2:** Modify the CNN architecture to improve accuracy.

📌 **Task 3:** Implement **transfer learning** using pre-trained models (e.g., VGG16, ResNet).

📜 SUMMARY

✓ **CNNs** are the backbone of modern **image recognition and computer vision**.
✓ **Convolutional Layers detect patterns**, while **Pooling Layers reduce complexity**.
✓ **ReLU activation** speeds up training by adding non-linearity.
✓ CNNs power **self-driving cars, medical imaging, security systems, and more**.

📌 ASSIGNMENT 1:

☑ IMPLEMENT A DIGIT RECOGNITION SYSTEM USING CNN.

📌 Assignment Solution 1: Implement a Digit Recognition System using CNN

🎯 Objective:

The goal of this assignment is to **implement a Convolutional Neural Network (CNN)** to recognize handwritten digits using the **MNIST dataset**. This dataset consists of **28×28 grayscale images of digits (0-9),** commonly used for image classification tasks.

By completing this assignment, you will learn how to:

✅ Load and preprocess image datasets for deep learning.
✅ Build a CNN model using **TensorFlow & Keras**.
✅ Train and evaluate the model for digit classification.
✅ Visualize predictions and improve model accuracy.

🛠 Step 1: Install Required Libraries

Before starting, install TensorFlow and other dependencies:

```
!pip install tensorflow numpy matplotlib
```

✅ **TensorFlow:** Used for deep learning and model building.
✅ **NumPy:** For numerical computations.
✅ **Matplotlib:** To visualize images and predictions.

📓 Step 2: Import Required Libraries

```
import tensorflow as tf

from tensorflow import keras
```

```
import numpy as np

import matplotlib.pyplot as plt
```

✓ **tensorflow** – Deep learning framework.
✓ **keras** – API for building CNN models.
✓ **numpy** – Handles data arrays.
✓ **matplotlib** – Visualizes images.

📥 Step 3: Load & Preprocess the Dataset

```
# Load the MNIST dataset

mnist = keras.datasets.mnist

(X_train, y_train), (X_test, y_test) = mnist.load_data()


# Normalize the image pixel values to the range [0,1]

X_train, X_test = X_train / 255.0, X_test / 255.0


# Reshape data to fit CNN input format

X_train = X_train.reshape(-1, 28, 28, 1)

X_test = X_test.reshape(-1, 28, 28, 1)
```

✓ **Normalization:** Scales pixel values (0-255) to (0-1) for faster learning.
✓ **Reshaping:** Converts images into 4D tensors **(batch, height, width, channels)** for CNN input.

🔧 Step 4: Build the CNN Model

```
# Define the CNN model
model = keras.Sequential([
  keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
  keras.layers.MaxPooling2D(2,2),
  keras.layers.Conv2D(64, (3,3), activation='relu'),
  keras.layers.MaxPooling2D(2,2),
  keras.layers.Flatten(),
  keras.layers.Dense(128, activation='relu'),
  keras.layers.Dense(10, activation='softmax')
])

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

# Model Summary
model.summary()
```

✓ **Conv2D(32, (3,3), activation='relu')** – Extracts features like edges, shapes.

✓ **MaxPooling2D(2,2)** – Reduces feature map size to prevent overfitting.
✓ **Flatten()** – Converts 2D feature maps into 1D array.
✓ **Dense(128, activation='relu')** – Fully connected layer to process features.
✓ **Dense(10, activation='softmax')** – Classifies images into **10 digits (0-9)**.

🎯 Step 5: Train the CNN Model

```
# Train the model

model.fit(X_train, y_train, epochs=5, validation_data=(X_test, y_test))
```

✓ **Epochs = 5**: The model will go through the dataset 5 times.
✓ **Validation Data**: Evaluates model accuracy after each epoch.

📊 Step 6: Evaluate the Model

```
# Evaluate on test data

test_loss, test_acc = model.evaluate(X_test, y_test)

print(f"Test Accuracy: {test_acc:.2f}")
```

✓ **Measures the model's performance on unseen test images.**
✓ **Displays accuracy score** (should be **above 98%**).

📷 Step 7: Visualizing Model Predictions

```
# Make predictions on test data
```

```
predictions = model.predict(X_test)


# Display first 5 test images and their predicted labels
for i in range(5):
    plt.imshow(X_test[i].reshape(28,28), cmap='gray')
    plt.title(f"Predicted: {np.argmax(predictions[i])}, Actual: {y_test[i]}")
    plt.show()
```

✓ **Displays test images with their predicted and actual labels.**
✓ **Uses np.argmax() to get the class with highest probability.**

🛠 Step 8: Improving Model Performance

To further **improve accuracy**, try these optimizations:

- 🔹 **Increase Epochs:** Train for more epochs (e.g., epochs=10).
- 🔹 **Add Dropout Layers:** Prevents overfitting by randomly disabling neurons.
- 🔹 **Use Data Augmentation:** Applies transformations (rotation, flipping) to images.
- 🔹 **Use Pretrained Models:** Use CNN architectures like **VGG16, ResNet** for better results.

📌 Conclusion

✅ We successfully built a **CNN-based Digit Recognition System**.
✅ The model learned to recognize digits **0-9** from the MNIST

dataset.

✅ Achieved **high accuracy (~98%)** with a simple architecture.

✅ Implemented **model visualization** to analyze predictions.

💡 **Next Steps:**

✓ Apply the same CNN model to **other datasets** (e.g., Fashion MNIST).

✓ Experiment with **deeper networks** for more complex problems.

✓ Deploy the model as a **web or mobile application**.

📌 ASSIGNMENT 2:

☑ TRAIN A NEURAL NETWORK FOR SENTIMENT ANALYSIS ON MOVIE REVIEWS.

📌 Assignment Solution 2: Train a Neural Network for Sentiment Analysis on Movie Reviews

🎯 Objective

In this assignment, we will train a **Neural Network for Sentiment Analysis** on a dataset of movie reviews. Sentiment analysis is the process of using **Natural Language Processing (NLP) and Deep Learning** to determine whether a text expresses **positive or negative sentiment**.

By the end of this guide, you will:

✅ Understand **how sentiment analysis works**.
✅ Learn **how to preprocess text data** for deep learning.
✅ Train a **Neural Network using TensorFlow and Keras**.
✅ Evaluate the model's performance on test data.

🛠 Step 1: Install & Import Required Libraries

First, install and import the required Python libraries.

```
# Install required libraries (if not already installed)

!pip install tensorflow numpy pandas matplotlib scikit-learn nltk


# Import necessary libraries

import numpy as np

import pandas as pd
```

```
import tensorflow as tf

from tensorflow import keras

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import
pad_sequences

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, LSTM, Dense,
Dropout

from sklearn.model_selection import train_test_split

import nltk

from nltk.corpus import stopwords

import re
```

✓ Why these libraries?

- **TensorFlow & Keras:** To build and train the deep learning model.
- **NumPy & Pandas:** To handle numerical and tabular data.
- **Scikit-learn:** For data preprocessing and evaluation.
- **NLTK:** For text processing (stopword removal, tokenization).

🛠 Step 2: Load the Dataset

For this task, we will use the **IMDb movie reviews dataset**, which consists of **50,000 reviews labeled as positive (1) or negative (0)**.

```
# Load IMDb dataset from Keras

(X_train, y_train), (X_test, y_test) =
keras.datasets.imdb.load_data(num_words=10000)


# Print dataset shape

print(f"Training samples: {len(X_train)}, Testing samples:
{len(X_test)}")
```

✓ Understanding the Dataset:

- The dataset contains **50,000 movie reviews** labeled as **positive (1) or negative (0)**.
- Words are already tokenized into **integer indices**.
- We limit vocabulary size to **10,000 words** for efficiency.

🛠 Step 3: Preprocess the Data

Before feeding data into the model, we **preprocess and prepare** the text sequences.

```
# Set maximum sequence length (shorter reviews will be padded)

max_length = 200


# Pad sequences to ensure uniform input size

X_train = pad_sequences(X_train, maxlen=max_length,
padding='post', truncating='post')
```

```
X_test = pad_sequences(X_test, maxlen=max_length, padding='post', truncating='post')


# Print new shape after padding

print(f"Shape after padding: {X_train.shape}, {X_test.shape}")
```

✓ Why Padding?

- Movie reviews have **varying lengths**, so we **standardize** them to a fixed length (200 words).
- Padding ensures **all sequences have equal dimensions**, making them compatible for training.

🛠 Step 4: Build the Neural Network Model

We will now **build an LSTM (Long Short-Term Memory) neural network**, a type of **Recurrent Neural Network (RNN)** designed for sequence-based tasks like sentiment analysis.

```
# Define the model

model = Sequential([

  Embedding(input_dim=10000, output_dim=128, input_length=max_length),  # Word embeddings

  LSTM(64, return_sequences=True),  # LSTM layer with 64 units

  LSTM(32),  # Another LSTM layer with 32 units

  Dense(16, activation='relu'),  # Fully connected layer

  Dropout(0.5),  # Dropout for regularization
```

```
    Dense(1, activation='sigmoid')  # Output layer (binary classification)
])

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

# Display model summary
model.summary()
```

✓ **Model Architecture:**

- **Embedding Layer:** Converts words into dense vector representations.
- **LSTM Layers:** Capture sequential patterns in text.
- **Dense Layers:** Extract deeper patterns for classification.
- **Sigmoid Activation:** Outputs probabilities for binary sentiment classification.

🛠 Step 5: Train the Model

Now, we train the model using the training data.

```
# Train the model
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=5, batch_size=64)
```

✓ **Explanation:**

- **5 epochs** ensure the model gets enough training without overfitting.
- **Batch size of 64** optimizes training speed and stability.
- **Validation data** helps monitor performance.

🛠 Step 6: Evaluate Model Performance

After training, we evaluate the model on test data.

```
# Evaluate model on test set
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy:.2f}")
```

✓ **Interpretation:**

- Higher accuracy means better performance in distinguishing positive/negative reviews.
- If accuracy is low, **adjust hyperparameters or add more training data**.

🛠 Step 7: Making Predictions

Now, let's test the model on new reviews.

```
# Function to predict sentiment
def predict_review(review):
    tokenizer = Tokenizer(num_words=10000)
```

```
    sequence = tokenizer.texts_to_sequences([review])
    padded_sequence = pad_sequences(sequence, maxlen=max_length, padding='post', truncating='post')
    prediction = model.predict(padded_sequence)[0][0]

    if prediction >= 0.5:
        return "Positive Review 😊"
    else:
        return "Negative Review 😞"

# Example Reviews
print(predict_review("This movie was absolutely fantastic! I loved it."))
print(predict_review("It was a terrible movie, completely disappointing."))
```

✓ **Explanation:**

- **Text is tokenized and padded** before feeding into the model.
- If the model predicts **≥ 0.5 probability**, the review is **positive**.
- If the probability is **< 0.5**, the review is **negative**.
