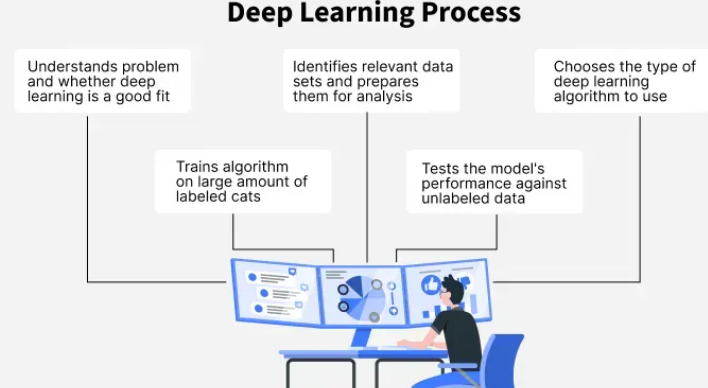
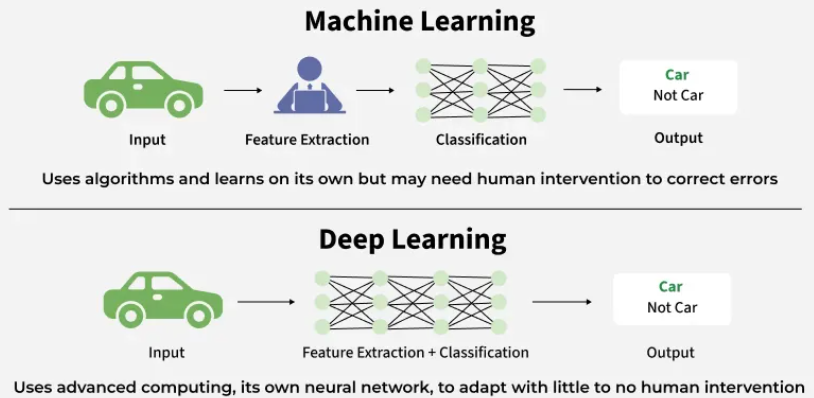
**INTRODUCTION TO DEEP LEARNING**

Deep Learning is transforming the way machines understand, learn and interact with complex data. Deep learning mimics neural networks of the human brain, it enables computers to autonomously uncover patterns and make informed decisions from vast amounts of unstructured data.





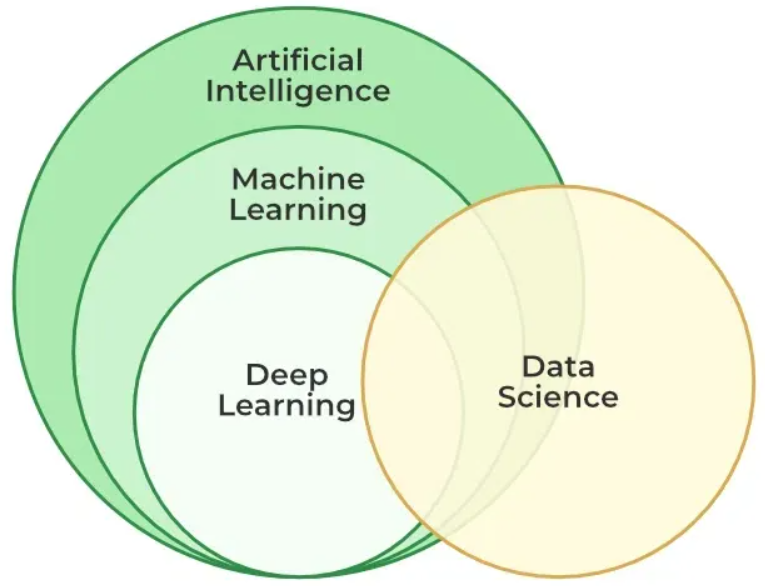
**How Deep Learning Works?**

Neural network consists of layers of interconnected nodes or neurons that collaborate to process input data. In a fully connected deep neural network data flows through multiple layers where each neuron performs nonlinear transformations, allowing the model to learn intricate representations of the data.

In a deep neural network the input layer receives data which passes through hidden layers that transform the data using nonlinear functions. The final output layer generates the model’s prediction.

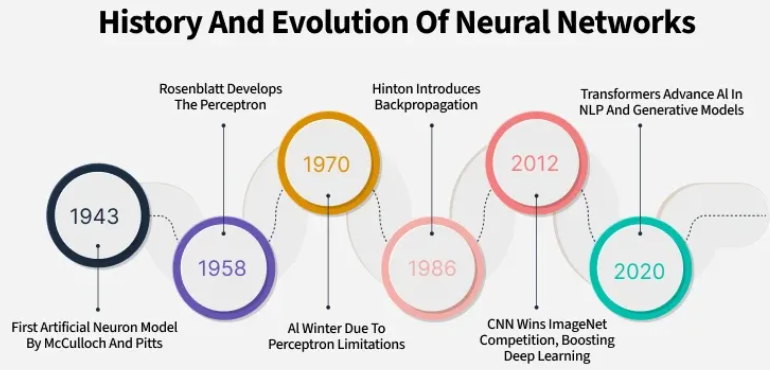
**Difference between Machine Learning and Deep Learning**

Machine learning and Deep Learning both are subsets of artificial intelligence but there are many similarities and differences between them.



**What is a Neural Network?**

Neural networks are machine learning models that mimic the complex functions of the human brain. These models consist of interconnected nodes or neurons that process data, learn patterns and enable tasks such as pattern recognition and decision-making.

****

Neural networks are capable of learning and identifying patterns directly from data without pre-defined rules. These networks are built from several key components:

* **Neurons**: The basic units that receive inputs, each neuron is governed by a threshold and an activation function.
* **Connections**: Links between neurons that carry information, regulated by weights and biases.
* **Weights and Biases**: These parameters determine the strength and influence of connections.
* **Propagation Functions**: Mechanisms that help process and transfer data across layers of neurons.
* **Learning Rule**: The method that adjusts weights and biases over time to improve accuracy.

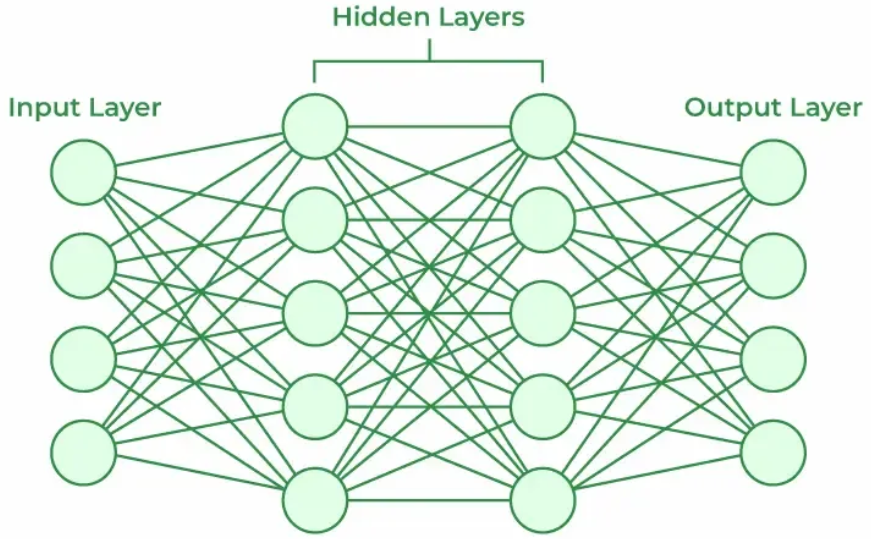
**Learning in neural networks follows a structured, three-stage process:**

1. **Input Computation**: Data is fed into the network.
2. **Output Generation**: Based on the current parameters, the network generates an output.
3. **Iterative Refinement**: The network refines its output by adjusting weights and biases, gradually improving its performance on diverse tasks.

**Importance of Neural Networks**

* **Identify Complex Patterns:** Recognize intricate structures and relationships in data; adapt to dynamic and changing environments.
* **Learn from Data:** Handle vast datasets efficiently; improve performance with experience and retraining.
* **Drive Key Technologies:** Power natural language processing (NLP); enable self-driving vehicles; support automated decision-making systems.
* **Boost Efficiency:** Streamline workflows and processes; enhance productivity across industries.
* **Backbone of AI:** Serve as the core driver of artificial intelligence progress; continue shaping the future of technology and innovation.

**Layers in Neural Network Architecture**

****

1. **Input Layer:** This is where the network receives its input data. Each input neuron in the layer corresponds to a feature in the input data.
2. **Hidden Layers:** These layers perform most of the computational heavy lifting. A neural network can have one or multiple hidden layers. Each layer consists of units (neurons) that transform the inputs into something that the output layer can use.
3. **Output Layer:** The final layer produces the output of the model. The format of these outputs varies depending on the specific task like classification, regression.

**Working of Neural Networks**

**1. Forward Propagation**

When data is input into the network, it passes through the network in the forward direction, from the input layer through the hidden layers to the output layer. This process is known as forward propagation. Here’s what happens during this phase:

**1. Linear Transformation:** Each neuron in a layer receives inputs which are multiplied by the weights associated with the connections. These products are summed together and a bias is added to the sum. This can be represented mathematically as:



**2. Activation:** The result of the linear transformation (denoted as ) is then passed through an activation function. The activation function is crucial because it introduces non-linearity into the system, enabling the network to learn more complex patterns. Popular activation functions include ReLU, sigmoid and tanh.

**2. Backpropagation**

After forward propagation, the network evaluates its performance using a loss function which measures the difference between the actual output and the predicted output. The goal of training is to minimize this loss. This is where backpropagation comes into play:

* **Loss Calculation:** The network calculates the loss which provides a measure of error in the predictions. The loss function could vary; common choices are mean squared error for regression tasks or cross-entropy loss for classification.
* **Gradient Calculation:** The network computes the gradients of the loss function with respect to each weight and bias in the network. This involves applying the chain rule of calculus to find out how much each part of the output error can be attributed to each weight and bias.
* **Weight Update:** Once the gradients are calculated, the weights and biases are updated using an optimization algorithm like stochastic gradient descent (SGD). The weights are adjusted in the opposite direction of the gradient to minimize the loss. The size of the step taken in each update is determined by the learning rate.

**3. Iteration**

This process of forward propagation, loss calculation, backpropagation and weight update is repeated for many iterations over the dataset. Over time, this iterative process reduces the loss and the network's predictions become more accurate.

Through these steps, neural networks can adapt their parameters to better approximate the relationships in the data, thereby improving their performance on tasks such as classification, regression or any other predictive modeling.

**Learning of a Neural Network**

**1. Learning with Supervised Learning**

In supervised learning, a neural network learns from labeled input-output pairs provided by a teacher. The network generates outputs based on inputs and by comparing these outputs to the known desired outputs, an error signal is created. The network iteratively adjusts its parameters to minimize errors until it reaches an acceptable performance level.

**2. Learning with Unsupervised Learning**

Unsupervised learning involves data without labeled output variables. The primary goal is to understand the underlying structure of the input data (X). Unlike supervised learning, there is no instructor to guide the process. Instead, the focus is on modeling data patterns and relationships, with techniques like clustering and association commonly used.

**3. Learning with Reinforcement Learning**

Reinforcement learning enables a neural network to learn through interaction with its environment. The network receives feedback in the form of rewards or penalties, guiding it to find an optimal policy or strategy that maximizes cumulative rewards over time. This approach is widely used in applications like gaming and decision-making.

**Types of Neural Networks**

There are seven types of neural networks that can be used.

* **Feedforward Networks:** It is a simple artificial neural network architecture in which data moves from input to output in a single direction.
* **Singlelayer Perceptron:**It has one layer and it applies weights, sums inputs and uses activation to produce output.
* **Multilayer Perceptron (MLP):** It is a type of feedforward neural network with three or more layers, including an input layer, one or more hidden layers and an output layer. It uses nonlinear activation functions.
* **Convolutional Neural Network (CNN):** It is designed for image processing. It uses convolutional layers to automatically learn features from input images, enabling effective image recognition and classification.
* **Recurrent Neural Network (RNN):**Handles sequential data using feedback loops to retain context over time.
* **Long Short-Term Memory (LSTM):**A type of RNN with memory cells and gates to handle long-term dependencies and avoid vanishing gradients.

**Advantages:**

Neural networks are widely used in many different applications because of their many benefits:

* **Adaptability:**Neural networks are useful for activities where the link between inputs and outputs is complex or not well defined because they can adapt to new situations and learn from data.
* **Pattern Recognition:** Their proficiency in pattern recognition renders them efficacious in tasks like as audio and image identification, natural language processing and other intricate data patterns.
* **Parallel Processing:**Because neural networks are capable of parallel processing by nature, they can process numerous jobs at once which speeds up and improves the efficiency of computations.
* **Non-Linearity:** Neural networks are able to model and comprehend complicated relationships in data by virtue of the non-linear activation functions found in neurons which overcome the drawbacks of linear models.

**Limitations:**

Neural networks while powerful, are not without drawbacks and difficulties:

* **Computational Intensity:**Large neural network training can be a laborious and computationally demanding process that demands a lot of computing power.
* **Black box Nature:**As "black box" models, neural networks pose a problem in important applications since it is difficult to understand how they make decisions.
* **Overfitting:** Overfitting is a phenomenon in which neural networks commit training material to memory rather than identifying patterns in the data. Although regularization approaches help to alleviate this, the problem still exists.
* **Need for Large datasets:**For efficient training, neural networks frequently need sizable, labeled datasets; otherwise, their performance may suffer from incomplete or skewed data.

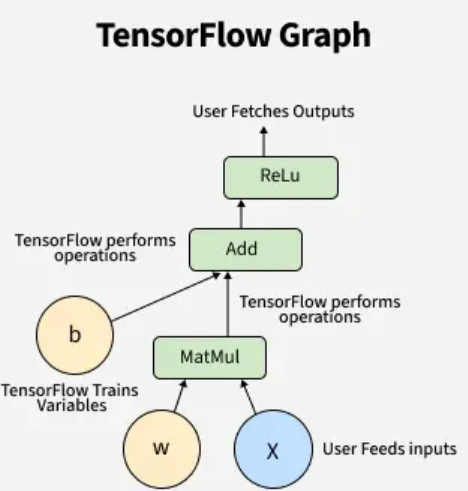
**Applications:**

Neural networks have numerous applications across various fields:

1. **Image and Video Recognition**: CNNs are extensively used in applications such as facial recognition, autonomous driving and medical image analysis.
2. **Natural Language Processing (NLP)**: RNNs and transformers power language translation, chatbots and sentiment analysis.
3. **Finance**: Predicting stock prices, fraud detection and risk management.
4. **Healthcare**: Neural networks assist in diagnosing diseases, analyzing medical images and personalizing treatment plans.
5. **Gaming and Autonomous Systems**: Neural networks enable real-time decision-making, enhancing user experience in video games and enabling autonomous systems like self-driving cars.

**Introduction to TensorFlow**

TensorFlow is an open-source framework for machine learning (ML) and artificial intelligence (AI) that was developed by Google Brain. It was designed to facilitate the development of machine learning models, particularly deep learning models by providing tools to build, train and deploy them across different platforms. It supports a wide range of applications from natural language processing (NLP) and computer vision (CV) to time series forecasting and reinforcement learning.



**Key Features**

**1. Scalability**

TensorFlow is designed to scale across a variety of platforms from desktops and servers to mobile devices and embedded systems. It supports distributed computing allowing models to be trained on large datasets efficiently.

**2. Comprehensive Ecosystem**

TensorFlow offers a broad set of tools and libraries including:

* **TensorFlow Core:** The base API for TensorFlow that allows users to define models, build computations and execute them.
* **Keras:** A high-level API for building neural networks that runs on top of TensorFlow, simplifying model development.
* **TensorFlow Lite:** A lightweight solution for deploying models on mobile and embedded devices.
* **TensorFlow.js:** A library for running machine learning models directly in the browser using JavaScript.
* **TensorFlow Extended (TFX):** A production-ready solution for deploying machine learning models in production environments.
* **TensorFlow Hub:** A repository of pre-trained models that can be easily integrated into applications.

**3. Automatic Differentiation (Autograd)**

TensorFlow automatically calculates gradients for all trainable variables in the model which simplifies the backpropagation process during training. This is a core feature that enables efficient model optimization using techniques like gradient descent.

**4. Multi-language Support**

TensorFlow is primarily designed for Python but it also provides APIs for other languages like C++, Java and JavaScript making it accessible to developers with different programming backgrounds.

**5. TensorFlow Serving and TensorFlow Model Optimization**

TensorFlow includes tools for serving machine learning models in production environments and optimizing them for inference allowing for lower latency and higher efficiency.

**TensorFlow Architecture**

The architecture of TensorFlow revolves around the concept of a computational graph which is a network of nodes (operations) and edges (data). Here's a breakdown of key components:

* **Tensors:** Tensors are the fundamental units of data in TensorFlow. They are multi-dimensional arrays or matrices used for storing data. A tensor can have one dimension (vector), two dimensions (matrix) or more dimensions.
* **Graph:** A TensorFlow graph represents a computation as a flow of tensors through a series of operations. Each operation in the graph performs a specific mathematical function on the input tensors such as matrix multiplication, addition or activation.
* **Session:** A session in TensorFlow runs the computation defined by the graph and evaluates the tensors. This is where the actual execution of the model happens enabling the training and inference processes.

**Keras Tutorial**

Keras high-level neural networks APIs that provide easy and efficient design and training of deep learning models. It is built on top of powerful frameworks like TensorFlow, making it both highly flexible and accessible. Keras has a simple and user-friendly interface, making it ideal for both beginners and experts in deep learning.



Keras simplifies the process of building and training deep learning models while abstracting away complex underlying operations. This tutorial covers everything you need to know to get started with Keras, from installation to advanced topics, making it a perfect guide for those looking to dive into deep learning

**How to Build a Model in Keras?**

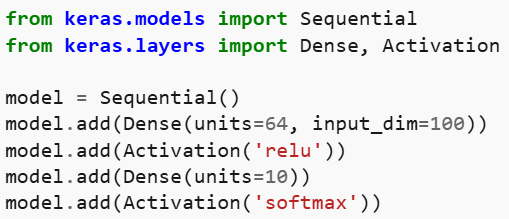
Keras provides two main ways to build models:

**1. Building Model using Sequential API**

The Sequential API are easy to work with models with a single input and output and a linear stack of layers

Here’s how you can define a Sequential model:

* We create a Sequential model.
* Add a fully connected (Dense) layer with 64 units and ReLU activation.
* Add another Dense layer with 10 units (for classification) and a Softmax activation.

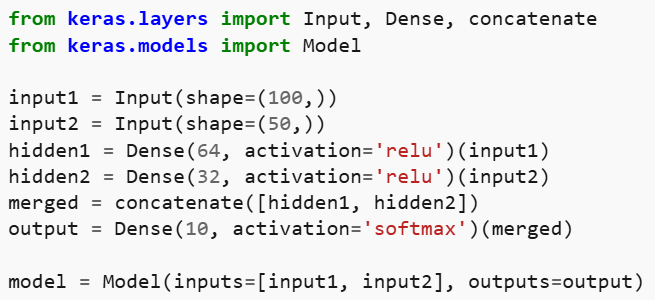


**2. Building Model using Functional API**

Functional API allows more flexibility in creating complex architectures. You can create models with shared layers, multiple inputs/outputs and skip connections.

**For example:**

* We define two input layers (input1 and input2).
* Create separate hidden layers for each input.
* Merge the hidden layers using the concatenate function.
* Finally, add an output layer with SoftMax activation.

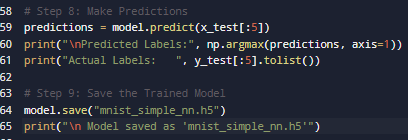


**Applications**

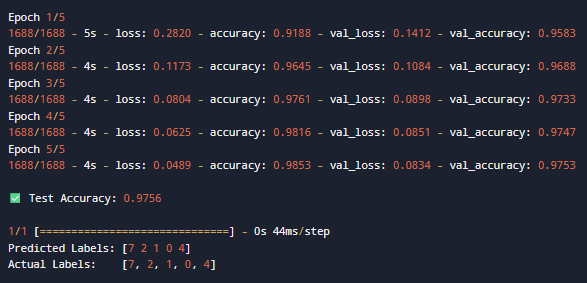
* **Image and Video Processing:**Keras facilitates tasks like image classification, object detection and video analysis through convolutional neural networks (CNNs). This makes it ideal for applications from medical imaging diagnostics to automated manufacturing quality control.
* **Natural Language Processing (NLP):**In NLP, Keras aids in building models for sentiment analysis and machine translation. Its support for sequential data processing is essential for systems capable of summarizing texts.
* **Time Series Forecasting:**Keras models equipped with LSTM or GRU layers are perfect for predicting time series data, which is used in fields like finance for stock price predictions or meteorology for weather forecasting.
* **Autonomous Systems:**Keras helps process real-time data from sensors in robotics and autonomous vehicles, It eases complex decision-making processes necessary for task performance without human input.

**Simple Neural Network Using Keras (MNIST Digit Classification)**

****

****

**OUTPUT:**

****

**EXPLANATION:**

* 1. **Import Required Libraries**
* We use **TensorFlow** and **Keras** to build and train the neural network.
* layers helps in adding hidden layers easily.
* NumPy is used for handling numerical data and arrays.
  1. **Load the MNIST Dataset**
* The MNIST dataset is included in Keras.
* It contains **70,000 grayscale images** of handwritten digits (0–9).
  + **60,000** → training data
  + **10,000** → testing data
* Each image is **28×28 pixels**.
  1. **Normalize the Data**
* Image pixel values range from 0 to 255.
* Dividing by 255 converts them into a **range between 0 and 1**.
* This improves training speed and accuracy.

**4. Flatten the Images**

* Each 28×28 image is reshaped into a **784-dimensional vector**.
* Neural networks require 1D feature vectors as input (not 2D images).

**5. Build the Neural Network Architecture**

* We use a **Sequential model** (layer-by-layer).
* Layers:
  1. **Dense(128, activation='relu')** – first hidden layer with 128 neurons.
  2. **Dense(64, activation='relu')** – second hidden layer with 64 neurons.
  3. **Dense(10, activation='softmax')** – output layer with 10 neurons for 10 digits.
* **ReLU (Rectified Linear Unit)** activation helps the model learn complex features.
* **Softmax** activation converts outputs into probabilities for classification.

**6. Compile the Model**

* **Optimizer:** Adam – adaptive optimization algorithm for faster convergence.
* **Loss Function:** sparse\_categorical\_crossentropy – suitable for multi-class classification with integer labels.
* **Metrics:** accuracy – measures prediction correctness during training.

**7. Train the Model**

* We train the model using:
* model.fit(x\_train, y\_train, epochs=5, batch\_size=32, validation\_split=0.1)
* **Epochs = 5:** Model goes through the dataset five times.
* **Batch size = 32:** Updates weights after processing 32 samples.
* **Validation split = 0.1:** Uses 10% of training data to validate accuracy.
* During training, the model continuously improves its performance.

**8. Evaluate the Model**

* The model is tested using the unseen **x\_test** and **y\_test** datasets.
* We get:
  + **Test Loss:** how far predictions are from actual values.
  + **Test Accuracy:** percentage of correct predictions.

**9. Make Predictions**

* The model predicts labels for unseen images:

**INTERPRETATION:**

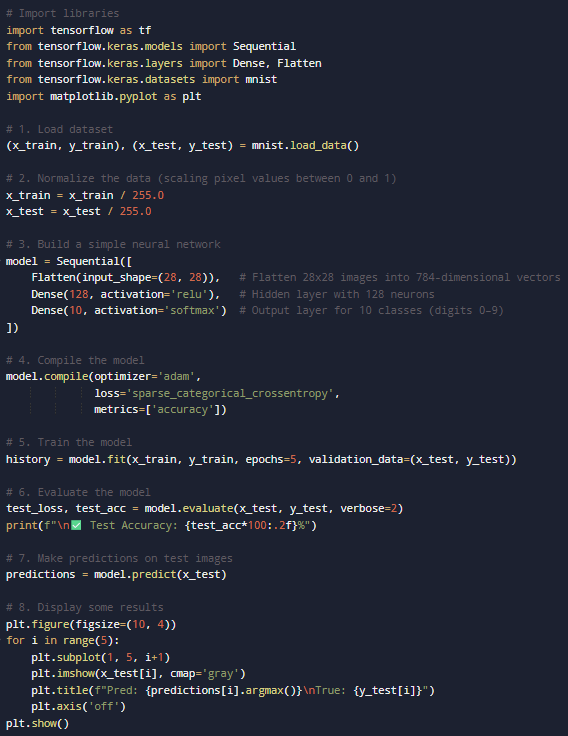
* The neural network **learned to recognize handwritten digits** with about **97–98% accuracy**.
* The model generalizes well (similar validation and test accuracies).
* You can reload the model anytime using:
* This saved .h5 model can be reused for predictions, further training, or deployment.
* model = keras.models.load\_model("mnist\_simple\_nn.h5")

**Image Classification using a Simple Neural Network (MNIST Dataset)**

**Project Overview:**

This project demonstrates how to build a **basic feed-forward neural network** to classify handwritten digits (0–9) from the **MNIST dataset**. The dataset contains **70,000 grayscale images** (28x28 pixels) of handwritten digits — 60,000 for training and 10,000 for testing.

Each image is labeled with the correct digit. The model learns to recognize patterns such as edges, loops, and curves associated with each digit.





**Overview**

* This project demonstrates how a simple **neural network** can classify handwritten digits (0–9) using the **MNIST dataset**.
* The **MNIST dataset** consists of **70,000 grayscale images** of handwritten digits — 60,000 for training and 10,000 for testing.
* Each image is **28×28 pixels** and labeled with the correct digit.
* The goal is to train the model to recognize and correctly classify new, unseen images.

**Libraries Used**

* **TensorFlow** – Deep learning library for building and training neural networks.
* **Keras (within TensorFlow)** – High-level API for easy model creation and training.
* **Matplotlib** – Visualization library for displaying images and prediction results.
* **MNIST Dataset** – Built-in dataset in Keras for handwritten digit recognition.

**Dataset Loading**

* The dataset is automatically downloaded from Keras datasets.
* It is split into:
  + **Training Data:** 60,000 images used for learning.
  + **Testing Data:** 10,000 images used for evaluation.
* Each image is represented as a **28×28 array** of pixel values between 0 and 255.

**Data Normalization**

* Pixel values are divided by 255 to convert the range from **0–255 → 0–1**.
* This process, known as **normalization**, ensures:
  + Faster model training.
  + More stable gradient updates.
  + Better overall accuracy.

**Neural Network Architecture**

* The model is built using **Sequential architecture**, which adds layers in sequence.
* It consists of three main layers:

**a. Flatten Layer**

* + Converts 2D images (28×28) into 1D arrays of 784 values.
  + Makes the data suitable for input into dense layers.

**b. Dense Layer (Hidden Layer)**

* + Contains **128 neurons**.
  + Uses the **ReLU (Rectified Linear Unit)** activation function.
  + Learns complex patterns and features from the input data.

**c. Dense Layer (Output Layer)**

* + Contains **10 neurons**, one for each digit (0–9).
  + Uses the **Softmax activation function**, which outputs probabilities for each class.
  + The class with the highest probability is taken as the final prediction.

**Model Compilation**

* The model is configured before training using the following parameters:
  + **Optimizer:** *Adam* – An adaptive optimization algorithm that efficiently updates weights.
  + **Loss Function:** *Sparse Categorical Crossentropy* – Measures how far predictions are from true labels.
  + **Metric:** *Accuracy* – Monitors the percentage of correct predictions during training and evaluation.

**Model Training**

* The training process involves feeding the model with training data multiple times (called **epochs**).
* In this project, the model is trained for **5 epochs**.
* After each epoch:
  + The model learns better representations of the digits.
  + Validation data (test set) is used to monitor progress and prevent overfitting.
* Training output shows:
  + **Loss:** The difference between predicted and actual outputs.
  + **Accuracy:** The proportion of correct predictions.

**Model Evaluation**

* After training, the model is tested on **10,000 unseen images**.
* The evaluation returns two metrics:
  + **Test Loss:** How well the model performs on unseen data.
  + **Test Accuracy:** Usually around **97–98%**, showing strong generalization performance.

**Making Predictions**

* The model predicts probabilities for each digit (0–9) for every test image.
* The digit with the highest probability is considered the predicted label.

**Visualization of Results**

* A few sample test images are displayed along with:
  + The **Predicted Digit**.
  + The **True Label**.
* This helps visually verify how well the model has learned.
* Typically, most predictions are correct, with only a few mismatches due to handwriting variation.

**Results Summary**

* **Training Accuracy:** ≈ 98%
* **Testing Accuracy:** ≈ 97–98%
* **Optimizer Used:** Adam
* **Loss Function:** Sparse Categorical Crossentropy
* **Model Type:** Feedforward Fully Connected Neural Network
* **Dataset Used:** MNIST (handwritten digits 0–9)

**Key Concepts Recap**

* **Neural Network:** A model inspired by the human brain that learns features from data.
* **Epoch:** One complete pass of the entire dataset through the network.
* **Activation Function:** Determines whether a neuron “fires” (ReLU and Softmax used here).
* **Loss Function:** Quantifies prediction errors.
* **Optimizer:** Adjusts weights to minimize loss and improve accuracy.
* **Accuracy Metric:** Measures model performance.

**Project Analysis:**

* **Training Accuracy:** ~98%
* **Test Accuracy:** ~97%
* **Loss Function:** Sparse Categorical Crossentropy (since labels are integers)
* **Optimizer:** Adam (Adaptive Moment Estimation)
* **Model Type:** Fully Connected Feedforward Neural Network

The model performs very well given its simplicity, achieving nearly **97–98% accuracy** on unseen data.

For more complex image tasks (like CIFAR-10 or ImageNet), **Convolutional Neural Networks (CNNs)** are preferred.