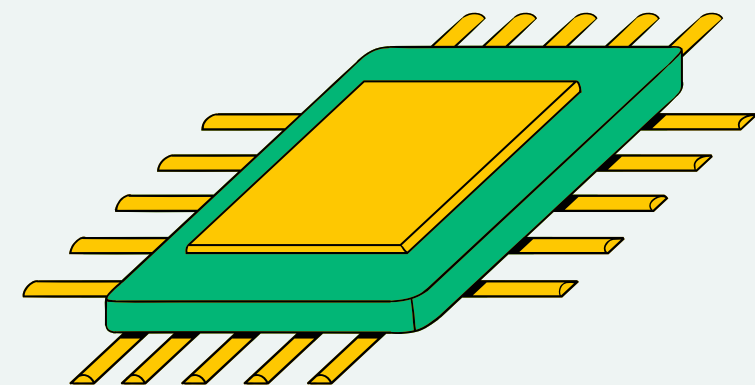


# DEEP LEARNING USING PYTORCH

## PRESENTATION

PRESENTED BY:

DR. R. DEVI





# PRESENTATION OUTLINE

- INTRODUCTION TO DEEP LEARNING
- INTRODUCTION TO PYTORCH
- TENSORS
- TENSOR OPERATIONS
- INTRODUCTION TO NEURAL NETWORKS
- SIMPLE ANN USING PYTORCH
- IMPLEMENTATION OF CNN USING PYTORCH



# DEEP LEARNING



Deep learning is a subset of machine learning, which is itself a subset of artificial intelligence (AI). It involves the use of neural networks with many layers (hence "deep") to model complex patterns in data. Unlike traditional machine learning algorithms, which may require feature engineering and manual intervention to work effectively, deep learning algorithms can automatically discover representations of data that are useful for tasks like classification, regression, and more.

## Key Concepts in Deep Learning:

**Neural Networks:** The foundation of deep learning, inspired by the human brain. They consist of layers of nodes (neurons) connected by edges (synapses).

**Layers:** Deep learning models typically have multiple layers, including an input layer, hidden layers, and an output layer. The depth of the network (number of layers) is a key factor in its ability to learn complex patterns.

**Activation Functions:** Functions applied to the output of each neuron to introduce non-linearity, enabling the network to learn more complex patterns. Common activation functions include ReLU (Rectified Linear Unit), Sigmoid, and Tanh.



- **Training:** Deep learning models are trained using large amounts of data. The process involves adjusting the weights of the connections between neurons to minimize the difference between the predicted and actual outputs (loss function).
- **Backpropagation:** A method used during training to update the weights in the network by calculating the gradient of the loss function with respect to each weight.
- **Learning Rate:** A hyperparameter that determines the step size at each iteration while moving toward a minimum of the loss function.



# APPLICATIONS

**Computer Vision**

**Natural Language Processing**

**Speech Recognition**

**Autonomous System**

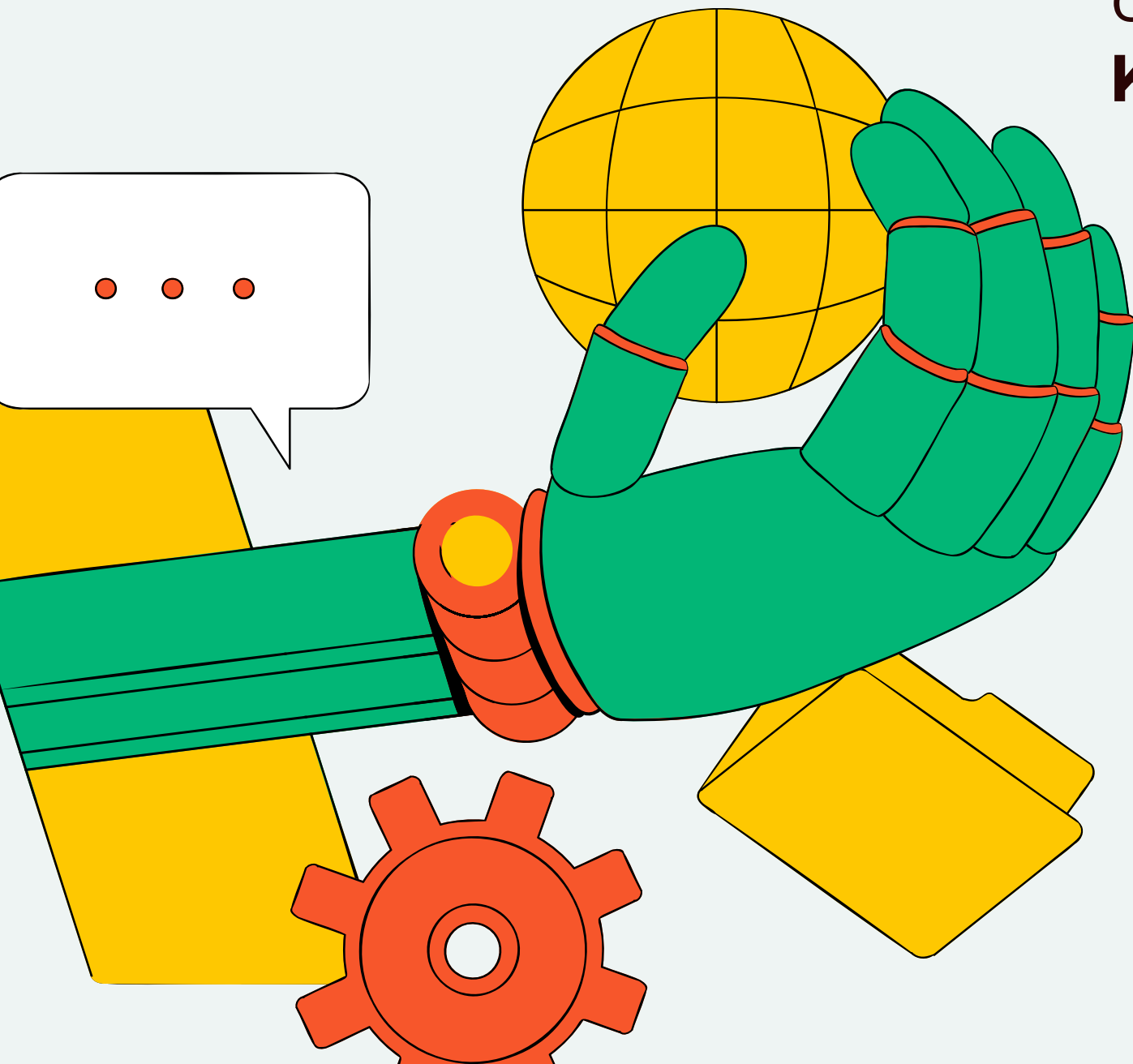


# PYTORCH

PyTorch is an open-source deep learning framework developed by Facebook's AI Research lab. It has become one of the most popular frameworks for building and training neural networks, especially in research and academia, due to its flexibility and ease of use.

## Key Features of PyTorch:


- **Dynamic Computation Graphs:** PyTorch uses a dynamic computation graph, also known as define-by-run. This means that the graph is built on-the-fly as operations are executed, allowing for greater flexibility and easier debugging compared to static graph frameworks like TensorFlow.
- **Tensors:** Tensors are the core data structure in PyTorch, similar to NumPy arrays but with the added capability of being used on GPUs for faster computation. PyTorch provides extensive support for tensor operations.

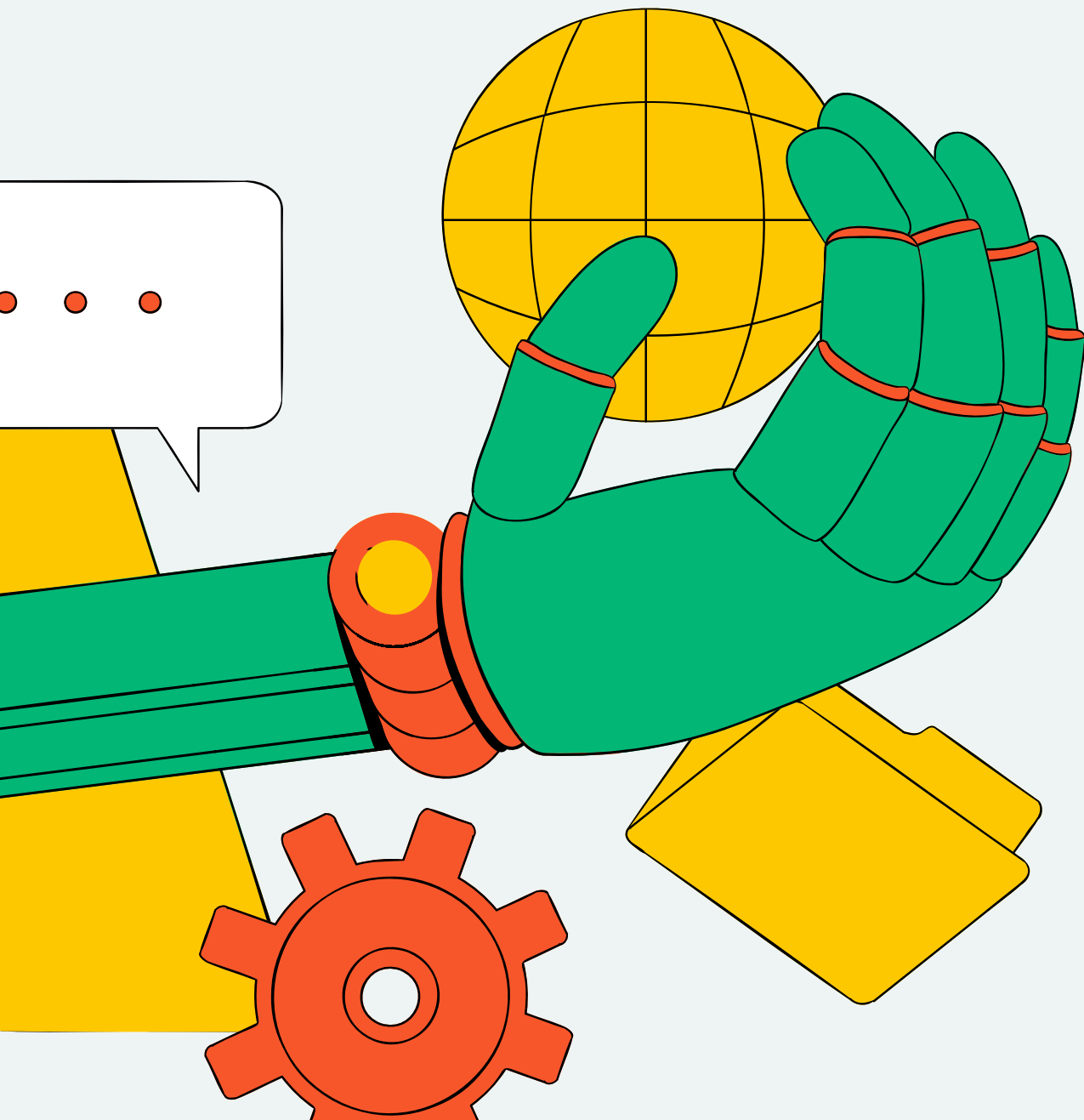




# PYTORCH

## Key Features of PyTorch:

- **Autograd:** PyTorch has an automatic differentiation engine called Autograd, which automatically calculates gradients needed for optimization. This makes it easier to implement backpropagation for training neural networks.
- **NN Module:** PyTorch provides a neural network module (`torch.nn`) that simplifies the creation of neural networks. It includes pre-built layers, loss functions, and activation functions that can be easily assembled to build complex models.
- **Optimizers:** PyTorch offers a variety of optimization algorithms through the `torch.optim` package, which can be used to adjust the model parameters based on the calculated gradients.
- **GPU Acceleration:** PyTorch can seamlessly  move computations from CPUs to GPUs, allowing for faster training of deep learning models.



# HOW PYTORCH WORKS?

- **Tensors:** PyTorch uses tensors, which are multi-dimensional arrays (like matrices) that store data. Tensors are the basic building blocks in PyTorch, and you can perform mathematical operations on them.
- **Neural Networks:** You can use PyTorch to build neural networks by stacking layers together. PyTorch makes it easy to define these networks and specify how they should process data.
- **Training Models:** Once you've built a model, PyTorch helps you train it using your data. It handles the heavy lifting, like optimizing the model to make better predictions. PyTorch is a popular tool for building and working with deep learning models. It's like a set of building blocks that helps developers and researchers create AI models without having to start from scratch.





# TENSORS

Tensor is a multi-dimensional matrix containing elements of a single data type. Similar To Numpy Arrays, but full of fun things that make them work better on GPU's (vs regular CPU's). default data type of float32. More suitable for deep learning than a numpy array.

- **import torch:** This command loads the PyTorch library, which is used for deep learning and tensor computations.
- **import numpy as np:** This imports NumPy, a library for numerical operations, often used for working with arrays and matrices in Python. The np is just a shorthand alias for easier usage.



# TENSORS

```
tensor_2d = torch.randn(3,4)
```

```
tensor_2d
```

```
#tensor_2d.dtype
```

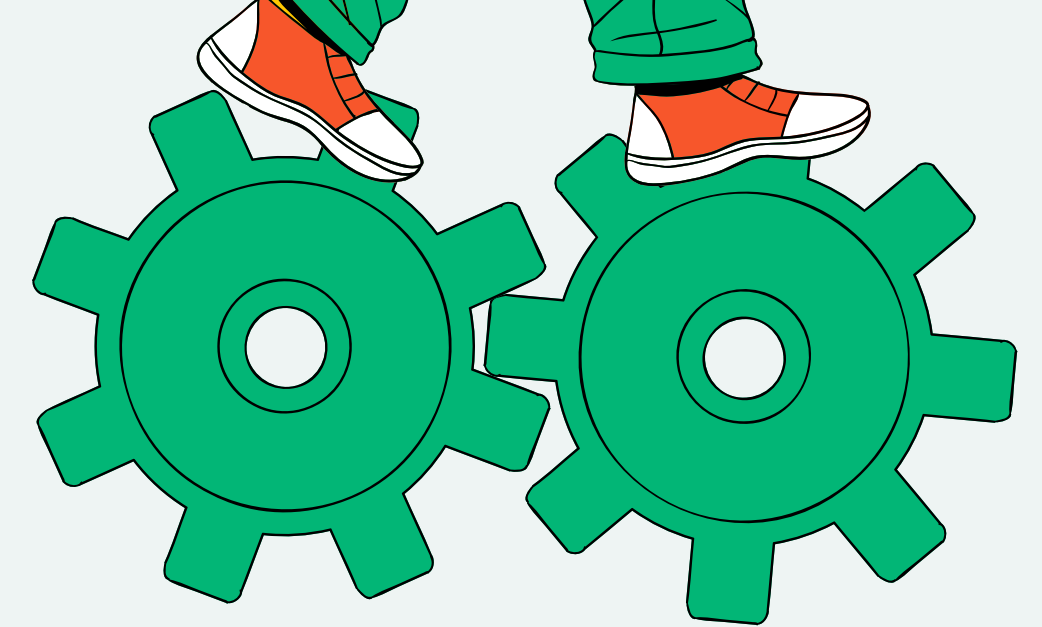
- `tensor_2d = torch.randn(3,4)` creates a 2D tensor of shape (3, 4) with random values drawn from a standard normal distribution (mean=0, variance=1).
- `tensor_2d.dtype` returns the data type of the elements in the tensor, which is typically `torch.float32` by default for tensors created with `torch.randn`.

```
## Create tensor out of numpy array
```

```
my_tensor = torch.tensor(np1)
```

```
my_tensor
```

You can create a PyTorch tensor from a NumPy array by passing the array into ``torch.tensor()``. This converts the NumPy array into a tensor, preserving the data and its structure.



# TENSOR OPERATIONS

## 1. Shape and View

### Shape:

- The shape of a tensor refers to the number of elements along each dimension of the tensor.
- For example, a tensor with shape (3, 2) has 3 rows and 2 columns.
- It is crucial to understand the shape of tensors when performing operations, as many operations require tensors to have compatible shapes.

### View:

- The view method in PyTorch allows you to reshape a tensor without changing its data.
- This is particularly useful when you need to change the shape of a tensor to fit a certain operation.
- Unlike the reshape method, view requires that the new shape is compatible with the original shape (i.e., the total number of elements must remain the same).



## Grabbing Items:

- You can grab single elements from a tensor using indexing, similar to how you would in a Python list or NumPy array.
- This is useful for extracting specific data points from your tensor.

## Slicing:

- Slicing in PyTorch is a technique to extract sub-tensors from a tensor.
- You can slice tensors using the : operator, similar to Python lists or NumPy arrays.

## Reshape

- Reshape refers to changing the dimensions of a tensor without altering its data.
- This operation is useful when you need to fit your data into a specific format required by a model or operation.



# TENSOR MATH OPERATIONS

## 1. Add

- Adds corresponding elements of two tensors of the same shape, or adds a scalar value to each element of a tensor.

## 2. Subtract

- Subtracts corresponding elements of two tensors of the same shape, or subtracts a scalar value from each element of a tensor.

## 3. Multiply

- Multiplies corresponding elements of two tensors of the same shape, or multiplies each element of a tensor by a scalar value.

## 4. Divide

- Divides corresponding elements of two tensors of the same shape, or divides each element of a tensor by a scalar value.

## 5. Remainders

- Computes the remainder of division between corresponding elements of two tensors of the same shape, or finds the remainder when each element of a tensor is divided by a scalar value.





# TENSOR MATH OPERATIONS

## 6. Exponents

- Raises each element of a tensor to the power of a given exponent, or computes the power of corresponding elements between two tensors.

## 7. Shorthand and Longhand

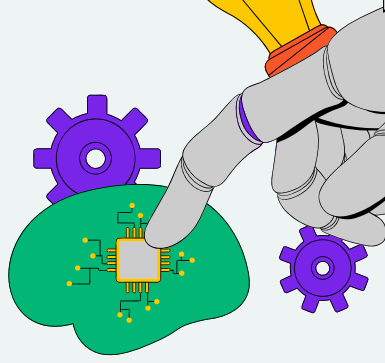
- Shorthand: Refers to the concise syntax for tensor operations, e.g., using `+`, `-`, `*`, and `/` for addition, subtraction, multiplication, and division.
- Longhand: Involves using explicit functions like `torch.add()`, `torch.subtract()`, `torch.multiply()`, and `torch.div()` for the same operations.

## 8. Reassignment

- Updates the value of an existing tensor by performing an in-place operation, such as using `+=` for addition or `*=` for multiplication.



# NEURAL NETWORKS

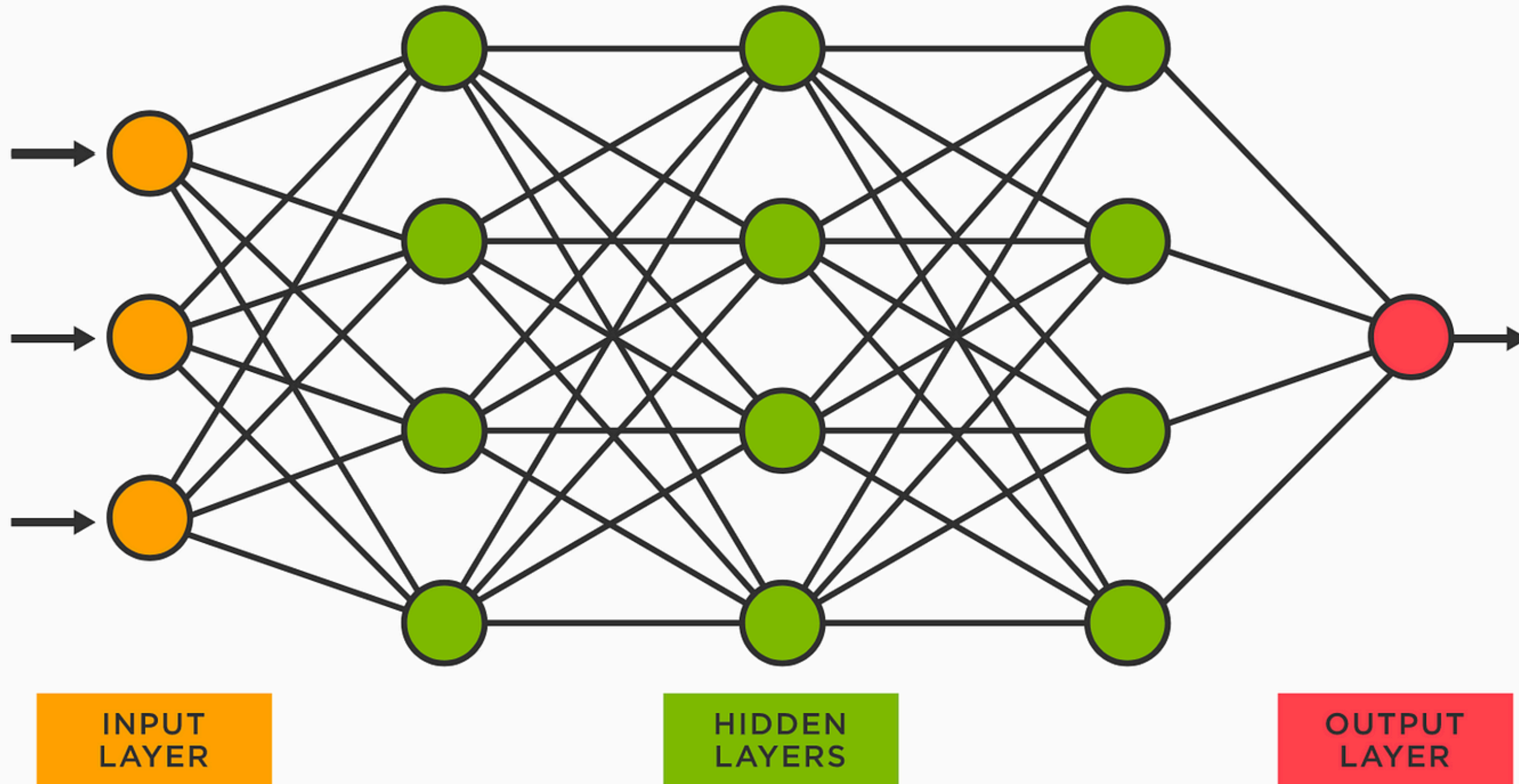


**Neural networks** are a type of machine learning model inspired by the human brain's structure and functioning. They consist of interconnected layers of nodes, or "neurons," each of which processes information and passes it to the next layer. Here's a quick overview:

1. **Input Layer:** This layer receives the raw data. Each node in this layer represents a feature of the input data.
2. **Hidden Layers:** These layers, which can be multiple, perform computations and transformations on the data. Each node in a hidden layer performs a weighted sum of the inputs, applies an activation function, and passes the result to the next layer.
3. **Output Layer:** This layer produces the final output of the network, such as a classification label or a numerical value.
4. **Weights and Biases:** Connections between nodes have associated weights that adjust during training to minimize the error in predictions. Each node also has a bias term.
5. **Activation Functions:** Functions applied to the output of each neuron to introduce non-linearity, allowing the network to learn complex patterns. Common examples include ReLU (Rectified Linear Unit), sigmoid, and tanh.

Neural networks are used in a variety of tasks, including image and speech recognition, natural language processing, and game playing, among others.

# NEURAL NETWORK



# SIMPLE ARTIFICIAL NEURAL NETWORK USING IRIS DATASET

## 1.Importing Libraries

- torch: PyTorch, a library for building neural networks.
- nn: Module in PyTorch to help build layers.
- F: Provides activation functions and other utilities.
- pandas: Used for data manipulation.
- matplotlib: Used for plotting graphs.
- %matplotlib inline: Ensures that plots are displayed in the notebook.

## 2.Creating the Neural Network Model

Model: A class that defines the architecture of our neural network.

- init: Initializes the layers.
  - fc1: First fully connected layer, connecting input to the first hidden layer.
  - fc2: Second fully connected layer, connecting the first hidden layer to the second hidden layer.
  - out: Output layer, connecting the second hidden layer to the output.
- forward: Defines the forward pass, where data passes through the network.
  - F.relu: Applies the ReLU activation function to the output of the layers.

### 3. Setting a Random Seed

`torch.manual_seed`: Ensures that the random numbers generated are the same every time the code runs, making the results reproducible.

### 4. Loading and Preparing the Dataset

- `url`: URL of the Iris dataset.
- `pd.read_csv`: Reads the CSV file into a DataFrame.
- `replace`: Converts the flower names into numerical labels (Setosa -> 0, Versicolor -> 1, Virginica -> 2).

### 5. Splitting Data into Features (X) and Labels (y)

- `X`: Contains the features (petal length, petal width, etc.).
- `y`: Contains the labels (flower types).



## **6. Converting Data to Numpy Arrays**

- Converts the DataFrame columns into numpy arrays for easy processing.

## **7. Splitting Data into Training and Testing Sets**

- `train_test_split`: Splits the data into training and testing sets (80% training, 20% testing).

## **8. Converting Data to Tensors**

- Converts numpy arrays into PyTorch tensors, the format required for input to the neural network.

## **9. Defining the Loss Function and Optimizer**

- `criterion`: Measures how far the predicted output is from the actual output.
- `optimizer`: Updates the model's parameters to reduce the loss. Here, the Adam optimizer is used with a learning rate of 0.01.

## 10. Training the Model

- `epochs`: Number of times the model sees the entire training set.
- `y_pred`: Predicted output for the training set.
- `loss`: The error between predicted and actual labels.
- `loss.backward`: Calculates the gradient of the loss.
- `optimizer.step`: Updates the model parameters.
- `losses.append`: Stores the loss for each epoch.

## 11. Plotting the Training Loss

`plt.plot`: Plots the loss over epochs to see how the model is improving.

## 12. Evaluating the Model on the Test Set

- `torch.no_grad()`: Disables gradient calculation during evaluation, speeding up the process.
- `y_eval`: Model's predictions on the test set.
- `loss`: Calculates the loss/error on the test set.

### 13. **Checking Model Accuracy**

- `correct`: Tracks the number of correct predictions.
- `y_val.argmax().item()`: Gets the class with the highest predicted probability.
- `Print statement`: Displays the prediction and whether it was correct.

### 14. **Making Predictions on New Data**

- `new_iris`: A tensor representing a new flower's features.
- `model(new_iris)`: Makes a prediction on this new flower.

### 15. **Saving and Loading the Model**

- `torch.save`: Saves the model's learned parameters.
- `load_state_dict`: Loads the saved parameters into a new model instance.
- `eval()`: Sets the model to evaluation mode, turning off certain layers like dropout that are only used during training.

# CNN

A Convolutional Neural Network (CNN) is a type of deep learning algorithm used primarily for processing and analyzing visual data. CNNs are particularly well-suited for image classification, object detection, and other tasks involving visual perception.

## **Key Components of a CNN:**

### **1. Convolutional Layers:**

- These layers apply a series of filters (also called kernels) to the input image or data. The filters move across the image to capture features such as edges, textures, and patterns. This process is known as convolution, which helps the network learn spatial hierarchies of features.

**2.Pooling Layers:**Pooling layers reduce the spatial dimensions (width and height) of the feature maps while retaining the most important information. The most common type is max pooling, which takes the maximum value from a patch of the feature map. This helps in making the network more efficient and less sensitive to small changes in the input.

**3.ReLU (Rectified Linear Unit) Activation:**After each convolution, an activation function like ReLU is applied to introduce non-linearity. ReLU sets all negative values in the feature map to zero, allowing the network to learn more complex patterns.

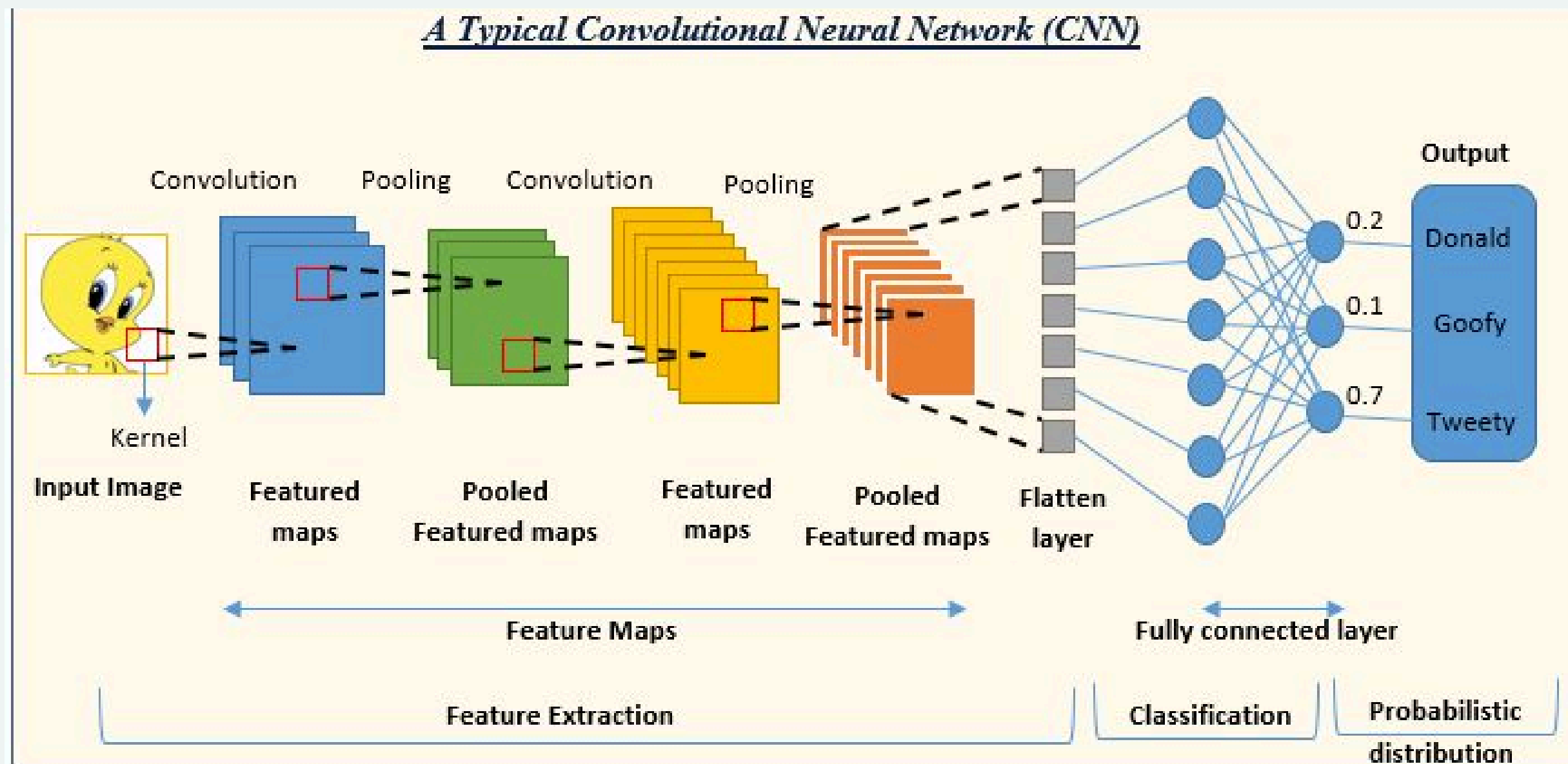
**4.Fully Connected Layers:**After the series of convolutional and pooling layers, the high-level features are flattened into a one-dimensional vector and passed through fully connected layers. These layers combine the features learned by the convolutional layers to make the final prediction, such as classifying an image into categories.

**5.Output Layer:**The output layer typically uses a softmax function for multi-class classification problems. It converts the final outputs into probabilities that sum to one, allowing the network to predict the class of the input image.



# How CNNs Work:

- **Input:** The CNN takes an image as input, usually in the form of a grid of pixel values.
- **Feature Extraction:** Through the convolutional layers, the CNN extracts various features, starting from low-level features like edges to more complex features like shapes and objects.
- **Classification:** After feature extraction, the network classifies the image into one or more categories based on the features.



# APPLICATIONS

**Image Recognition**

**Object Detection**

**Medical Imaging**

**Autonomous Vehicles**



# Implementation of a Convolutional Neural Network (CNN) using PyTorch for the MNIST dataset

## 1. Imports and Setup

- torch: The core library for working with tensors and neural networks in PyTorch.
- nn: Contains modules and classes for constructing and training neural networks.
- F: Functional interface for neural network layers and operations.
- DataLoader: A utility to load data in batches and shuffle them.
- datasets and transforms: Part of the torchvision package, which provides datasets and transformations for image data.
- make\_grid: Utility for visualizing batches of images.
- numpy, pandas, confusion\_matrix, matplotlib: Standard Python libraries for data manipulation and visualization.

## 2. Data Preparation

- `transform = transforms.ToTensor()`: Converts the MNIST images to PyTorch tensors, which are required for neural network training.
- `datasets.MNIST`: Loads the MNIST dataset from root directory, applying the specified transformation (`ToTensor`) to the images. `train=True` loads the training data, and `train=False` loads the test data.

## 3. DataLoader Setup

`DataLoader`: Wraps the dataset and allows easy iteration over data batches. `batch_size=10` specifies that each batch will contain 10 images. `shuffle=True` randomizes the order of the training data for each epoch.

## 4. CNN Model Definition

- `nn.Conv2d(1, 6, 3, 1)`: A convolutional layer with 1 input channel (grayscale image), 6 output channels (feature maps), a kernel size of 3x3, and a stride of 1. This layer will learn 6 different 3x3 filters to scan across the input image.
- `nn.Conv2d(6, 16, 3, 1)`: The second convolutional layer with 6 input channels (from previous layer), 16 output channels, and a 3x3 kernel size.

## 5. Single Image Convolution Example

- `X_Train.view(1,1,28,28)`: Reshapes the image to the format expected by the convolutional layer (batch size, channels, height, width).
- `F.relu(conv1(x))`: Applies the first convolution followed by a ReLU activation function, which introduces non-linearity.
- `F.max_pool2d(x,2,2)`: Applies a 2x2 max pooling operation, reducing the spatial dimensions by half (e.g., 26x26 -> 13x13).

## 6. Model Class Definition

`ConvolutionalNetwork`: The class defines a CNN with two convolutional layers followed by three fully connected layers.

- `conv1, conv2`: Convolutional layers as defined earlier.
- `fc1, fc2, fc3`: Fully connected layers that progressively reduce the dimensionality from 400 (1655) to 10 output classes.
- `forward`: Defines the forward pass of the model, which is how data flows through the network.



## 7. Model Instantiation

- `torch.manual_seed(41)`: Sets a random seed to ensure reproducibility.
- `model = ConvolutionalNetwork()`: Creates an instance of the CNN model.

## 8. Loss Function and Optimizer

- `criterion = nn.CrossEntropyLoss()`: Defines the loss function, which is `CrossEntropyLoss`. It combines `nn.LogSoftmax()` and `nn.NLLLoss()` in one single class.
- `optimizer = torch.optim.Adam(model.parameters(), lr=0.001)`: Uses the Adam optimizer to adjust the weights of the network based on the gradients computed during backpropagation. The learning rate is set to 0.001.

## 9. Training and Testing Loop

- `epochs = 5`: Specifies the number of times the entire dataset will pass through the model.
- Training Loop:
- `y_pred = model(X_train)`: Gets predictions from the model

- `loss = criterion(y_pred, y_train)`: Computes the loss by comparing the predictions to the true labels.
- `loss.backward()`: Computes gradients.
- `optimizer.step()`: Updates model parameters using the computed gradients.
- Testing Loop: Evaluates the model on the test set without updating the weights.

## 10. Plotting Loss and Accuracy

`plt.plot`: Plots the training and validation loss/accuracy across epochs.

## 11. Final Accuracy Check

`correct.item()/len(test_data)*100`: Computes the accuracy on the entire test dataset.

## 12. Image Visualization and Prediction

- `plt.imshow`: Visualizes the chosen image.
- `new_prediction = model(test_data[1978][0].view(1,1,28,28))`: Predicts the label of the selected image.
- `new_prediction.argmax()`: Returns the index of the highest value in the prediction, which corresponds to the predicted digit.