Q1. What is Deep Learning? Explain its evolution, significance, and how it differs from traditional Machine Learning.

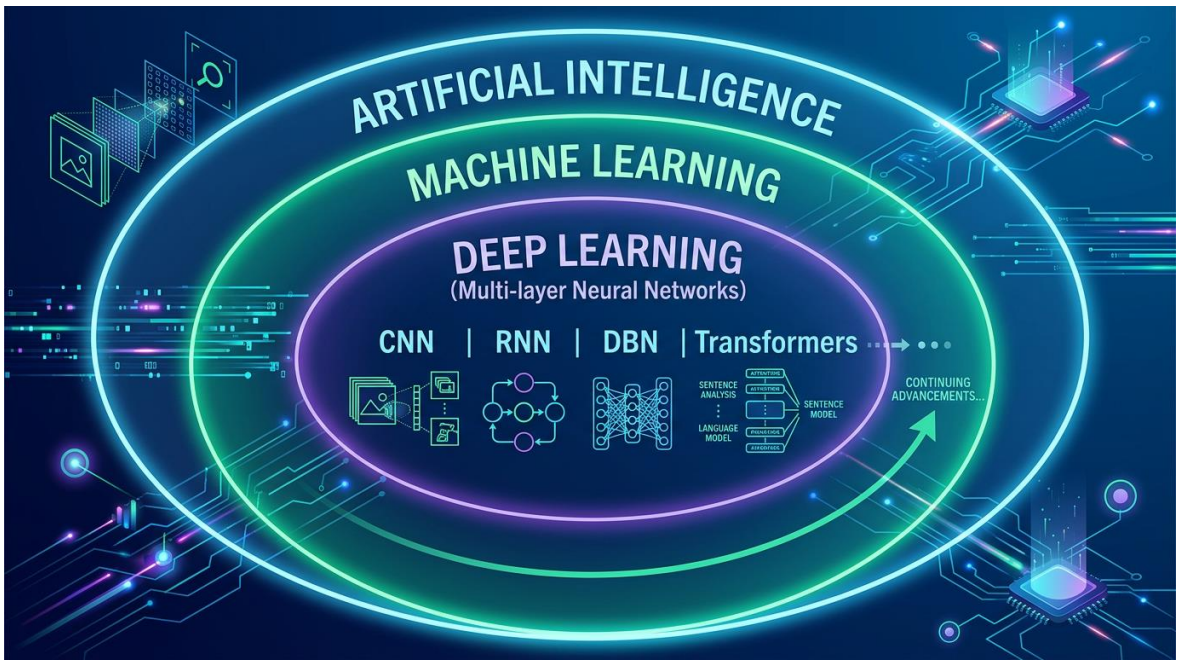
Answer:

1. Definition of Deep Learning

Deep Learning (DL) is a subfield of Artificial Intelligence (AI) and Machine Learning (ML) that uses artificial neural networks with multiple layers (hence 'deep') to automatically learn hierarchical representations of data. Each successive layer learns increasingly abstract features from the raw input, enabling the system to solve complex tasks such as image recognition, speech processing, and language understanding without explicit feature engineering.

Deep Learning	A class of ML algorithms using multi-layered neural networks to learn feature hierarchies directly from raw data, minimizing the need for manual feature engineering.

2. Position in the AI Hierarchy



3. Historical Evolution of Deep Learning

Year	Milestone
1943	McCulloch & Pitts propose the first mathematical model of a neuron
1958	Rosenblatt introduces the Perceptron — single-layer learning machine
1969	Minsky & Papert show limitations of single-layer perceptrons (XOR problem)
1986	Rumelhart, Hinton & Williams popularize Backpropagation algorithm
1989	LeCun et al. apply backprop to handwritten digit recognition

1997	Hochreiter & Schmidhuber introduce Long Short-Term Memory (LSTM)
1998	LeCun proposes LeNet-5 — first practical CNN for digit recognition
2006	Hinton et al. introduce Deep Belief Networks — revives deep learning
2012	AlexNet (Krizhevsky et al.) wins ImageNet — marks the 'DL revolution'
2014	GANs (Goodfellow), VGGNet, GoogLeNet introduced
2015	ResNet with 152 layers introduced; Batch Normalization
2017	Transformer architecture ('Attention is All You Need') introduced
2022+	ChatGPT, GPT-4, Gemini — large-scale generative AI

4. Traditional ML vs Deep Learning

Aspect	Traditional ML	Deep Learning
Feature Engineering	Manual — domain expert required	Automatic — learned from data
Data Requirement	Works on small/medium datasets	Requires large datasets (Big Data)
Performance	Plateau beyond certain data size	Continues to improve with more data
Compute	CPU-friendly	Requires GPU/TPU
Interpretability	Mostly interpretable	Black-box (limited explainability)
Examples	SVM, Decision Trees, Naive Bayes	CNN, RNN, Transformers, GANs
Architecture	Shallow (1-2 layers typical)	Deep (10 to 1000+ layers)

5. Why Deep Learning Works — Key Enablers

1. Big Data: Millions of labeled examples available (ImageNet, Common Crawl, etc.)
2. GPU Computing: Parallel matrix operations speed up training by 100x+
3. Improved Algorithms: ReLU, Batch Normalization, Dropout, Adam optimizer
4. Open Source Frameworks: TensorFlow, PyTorch made DL accessible
5. Pre-trained Models: Transfer learning allows reuse of trained knowledge

Key Takeaway

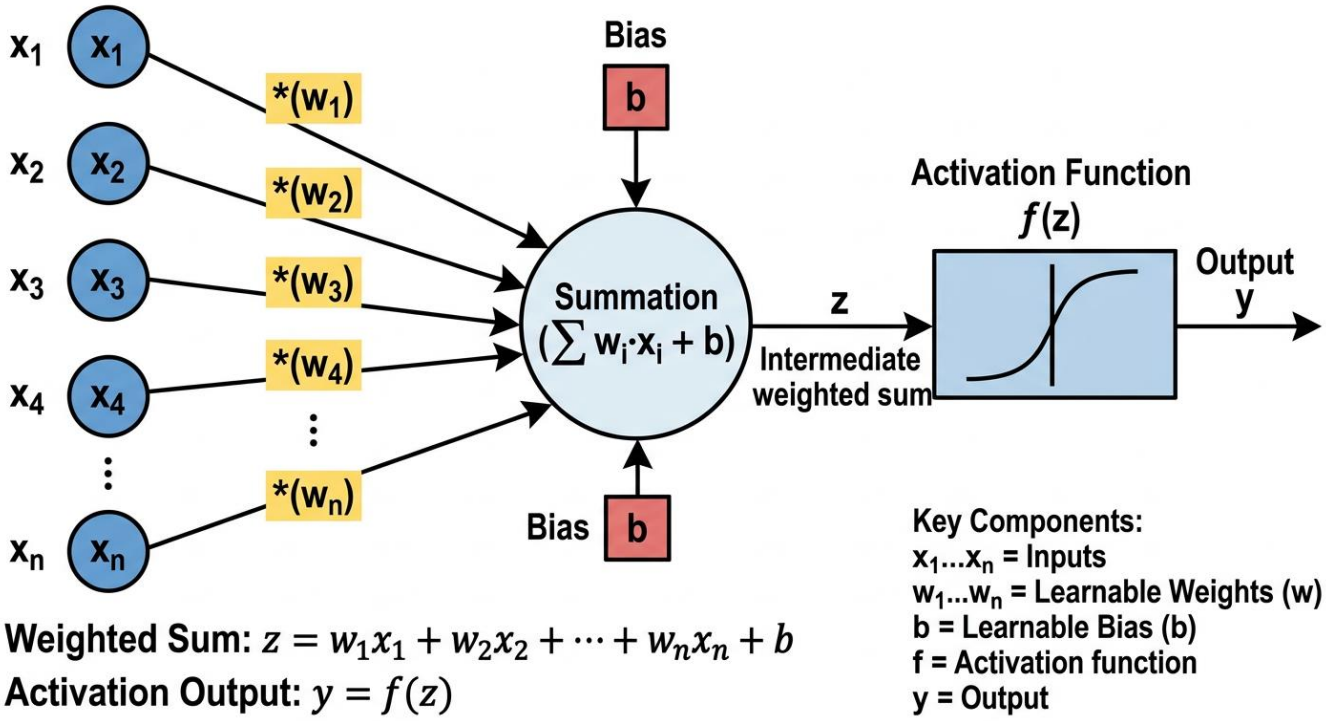
Deep Learning achieves 'end-to-end learning' — raw input to final output, Automatically discovering the optimal features at every stage. This eliminates the bottleneck of manual feature engineering that limited classical ML.

Q2. Describe in detail the building blocks of deep neural network architectures, including neurons, activation functions, loss functions, and back propagation.

Answer:

1. The Artificial Neuron (Perceptron)

The fundamental unit of any neural network is the artificial neuron, modeled after the biological neuron. It receives multiple inputs, applies weights, sums them up, adds a bias, and passes the result through an activation function to produce an output.

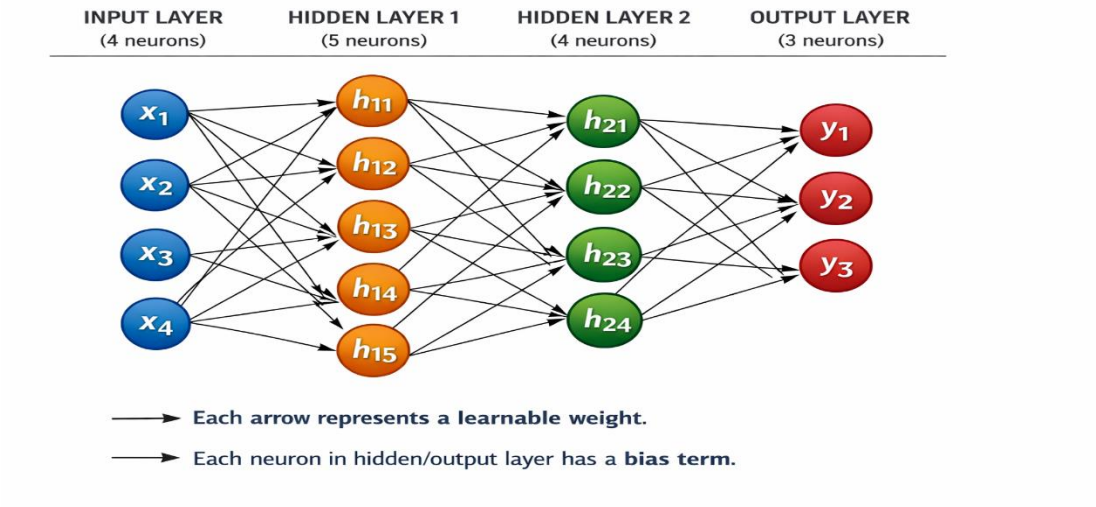


Weighted Sum: $z = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$

Activation Output: $y = f(z)$

2. Multi-Layer Perceptron (MLP) Architecture

Stacking neurons in layers creates a Multi-Layer Perceptron. Information flows from the input layer, through one or more hidden layers, to the output layer.



3. Activation Functions

Activation functions introduce non-linearity into the network, enabling it to learn complex, non-linear mappings from input to output. Without activation functions, stacking layers would be equivalent to a single linear transformation.

(a) Sigmoid Function

```
σ(x) = 1 / (1 + e^(-x))    Range: (0, 1)
```

Outputs values between 0 and 1. Historically used in hidden layers and output layers for binary classification. Suffers from the vanishing gradient problem for large/small inputs.

(b) Hyperbolic Tangent (Tanh)

```
tanh(x) = (e^x - e^(-x)) / (e^x + e^(-x))    Range: (-1, 1)
```

Zero-centered output makes it preferred over sigmoid for hidden layers. Still suffers from vanishing gradients at extremes.

(c) Rectified Linear Unit (ReLU)

```
f(x) = max(0, x)    Range: [0, +∞)
```

The most popular activation for hidden layers in deep networks. Computationally efficient, does not suffer from vanishing gradients for positive inputs. May cause 'dying ReLU' problem (neurons permanently output 0).

(d) Leaky ReLU & Parametric ReLU

```
Leaky ReLU: f(x) = max(αx, x),   α = 0.01 (small slope for x < 0)
```

Fixes dying ReLU by allowing a small gradient for negative inputs.

(e) Softmax

```
softmax(xi) = e^xi / Σj e^xj    Range: (0,1), sums to 1
```

Used in the output layer for multi-class classification. Converts raw scores (logits) into probabilities that sum to 1.

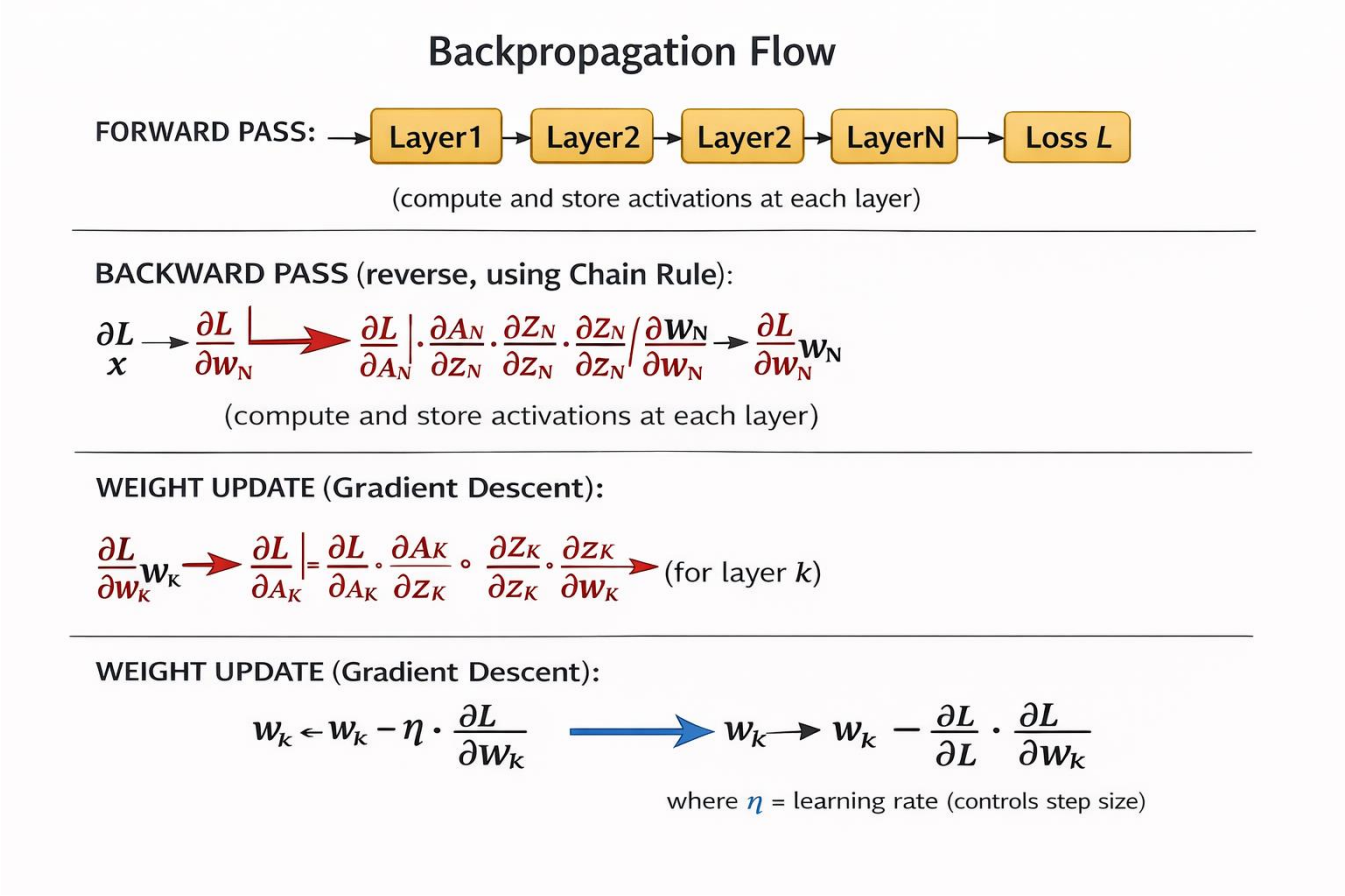
4. Loss Functions

The loss function measures the discrepancy between predicted outputs and true labels. Training minimizes this loss.

Loss Function	Formula / Use Case
Mean Squared Error (MSE)	L = (1/n) Σ(yi - ŷi)² | Regression tasks
Binary Cross-Entropy	L = -[y·log(ŷ) + (1-y)·log(1-ŷ)] | Binary classification
Categorical Cross-Entropy	L = -Σ yi·log(ŷi) | Multi-class classification
Hinge Loss	L = max(0, 1 - y·f(x)) | SVM-style classifiers

5. Backpropagation Algorithm

Backpropagation is the algorithm used to compute gradients of the loss with respect to all weights in the network, enabling gradient-based optimization. It applies the Chain Rule of calculus repeatedly from output to input layers.



6. Optimization Algorithms

Optimizer	Description
SGD (Stochastic GD)	Update weights using gradient of single sample; fast but noisy
Mini-Batch GD	Update using small batches (32-256); best balance of speed & stability
Momentum	Adds velocity term: v = βv - η∇L; reduces oscillations
Adam	Adaptive Moment Estimation; combines Momentum + RMSProp; most popular

Q3. Explain the Neocognitron model proposed by Fukushima. How does it relate to modern Deep Learning and Convolutional Neural Networks? (16 Marks)

Answer:

1. Introduction to Neocognitron

The Neocognitron is a hierarchical, multi-layered artificial neural network proposed by Kunihiko Fukushima in 1980. It was designed to achieve visual pattern recognition with position invariance — the ability to recognize patterns regardless of their location in the visual field. It is directly inspired by the mammalian visual cortex and serves as the biological and computational precursor to modern Convolutional Neural Networks (CNNs).

Neocognitron	A self-organizing neural network model (1980) that uses hierarchical layers of S-cells and C-cells to achieve shift-invariant visual pattern recognition, inspired by the mammalian visual cortex.

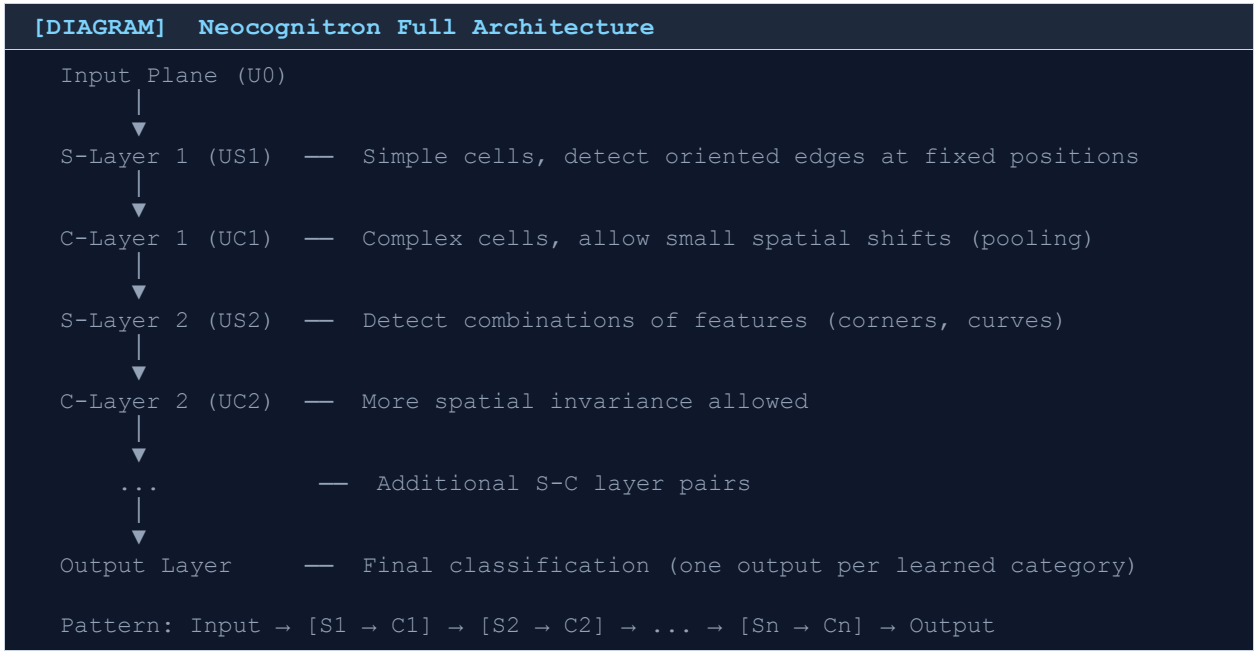
2. Biological Inspiration

Hubel and Wiesel (1959–1968) discovered that the mammalian visual cortex has two types of cells:

- **Simple Cells:** Respond to specific features (edges, orientations) at fixed positions.
- **Complex Cells:** Respond to features regardless of their exact position — spatial invariance.
- **Hypercomplex Cells:** Detect more abstract, combined features (corners, endpoints).

Fukushima modeled these as S-cells (Simple) and C-cells (Complex) in alternating layers.

3. Neocognitron Architecture



4. S-Cells and C-Cells Explained

S-Cells (Simple Cells)

S-cells act as feature detectors. Each S-cell responds to a specific local pattern (e.g., a vertical edge) in a small region of the previous layer's output. S-cells use a template-matching mechanism — they fire only when a specific combination of inputs from the previous C-layer matches the stored template. S-cells are analogous to convolutional filters in modern CNNs.

C-Cells (Complex Cells)

C-cells receive inputs from a group of S-cells detecting the same feature but at slightly different positions. A C-cell fires if any one of its input S-cells fires, achieving spatial tolerance (position invariance). This is analogous to Max Pooling in modern CNNs — taking the most active signal from a local region.

5. Key Properties of Neocognitron

6. Hierarchical Feature Extraction: Each layer learns more abstract features.
7. Shift Invariance: C-cells make recognition robust to small position changes.
8. Local Connectivity: S-cells connect only to a local region (receptive field), like convolutional kernels.
9. Unsupervised Learning: Original Neocognitron was trained unsupervised using competitive learning.
10. Multiple Feature Planes: Each layer has multiple feature maps for different feature types.

6. Neocognitron vs Modern CNN — Direct Mapping

Neocognitron Component	Modern CNN Equivalent	Function
S-cell layer	Convolutional Layer	Feature detection via learnable filters
C-cell layer	Pooling Layer (MaxPool/AvgPool)	Spatial invariance / downsampling
Template matching in S-cells	Convolution operation	Compute similarity of filter with input region
Receptive field	Kernel / Filter window	Local region of input a neuron responds to
Multiple feature planes per layer	Multiple feature maps (channels)	Detect multiple feature types simultaneously
Hierarchical S-C layer pairs	Conv-Pool blocks	Build increasingly abstract representations

7. Historical Significance

The Neocognitron was the first computational model to successfully demonstrate hierarchical feature learning for visual recognition. When Yann LeCun developed LeNet-5 (1998), he explicitly built on the Neocognitron's architecture, adding backpropagation-based supervised training. This lineage: Neocognitron → LeNet → AlexNet → ResNet forms the foundation of all modern computer vision systems.

Q4. Explain the architecture and working of Deep Convolutional Neural Networks (CNN) with all key components. Describe famous CNN architectures. (16 Marks)

Answer:

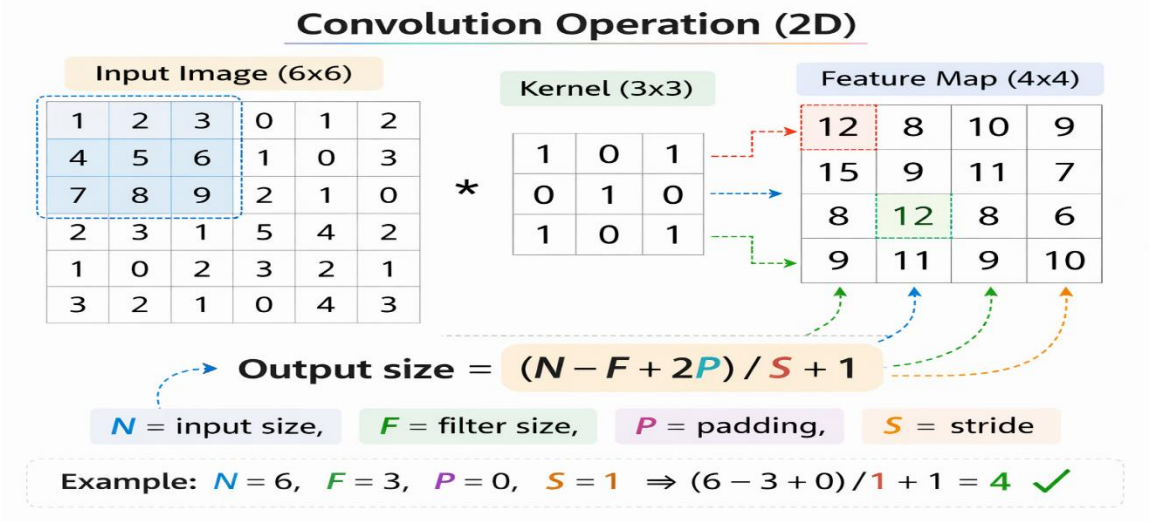
1. Introduction to CNN

A Convolutional Neural Network (CNN) is a specialized deep neural network designed primarily for processing grid-structured data such as images (2D grids of pixels) and videos (3D grids). CNNs exploit the spatial structure of input data through local connectivity, shared weights (parameter sharing), and pooling, making them highly efficient and effective for visual recognition tasks.

2. Core Components of a CNN

(a) Convolutional Layer — The Core Building Block

A convolutional layer applies learnable filters (kernels) to the input. Each filter slides across the input (convolution operation) producing a feature map that captures the presence of specific patterns (edges, textures, shapes).

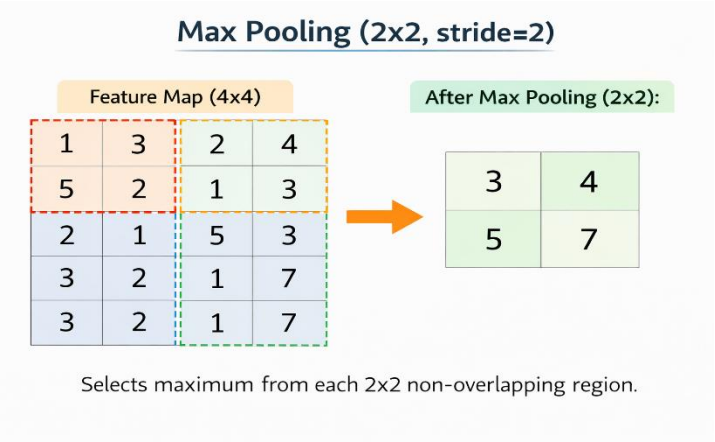


(b) Activation Layer (ReLU)

After each convolution, a non-linear activation (typically ReLU: f(x) = max(0,x)) is applied element-wise to the feature map. This introduces non-linearity, allowing the network to learn complex patterns.

(c) Pooling Layer

Pooling reduces the spatial dimensions of feature maps, decreasing computation and providing a degree of translation invariance. The two most common types are Max Pooling (takes the maximum value in each window) and Average Pooling.



(d) Batch Normalization

Normalizes the outputs of a layer to have zero mean and unit variance. Stabilizes and accelerates training, acts as a mild regularizer, and allows higher learning rates.

(e) Dropout

During training, randomly sets a fraction p (dropout rate) of neurons to zero. This prevents co-adaptation of neurons and acts as a powerful regularizer, significantly reducing overfitting.

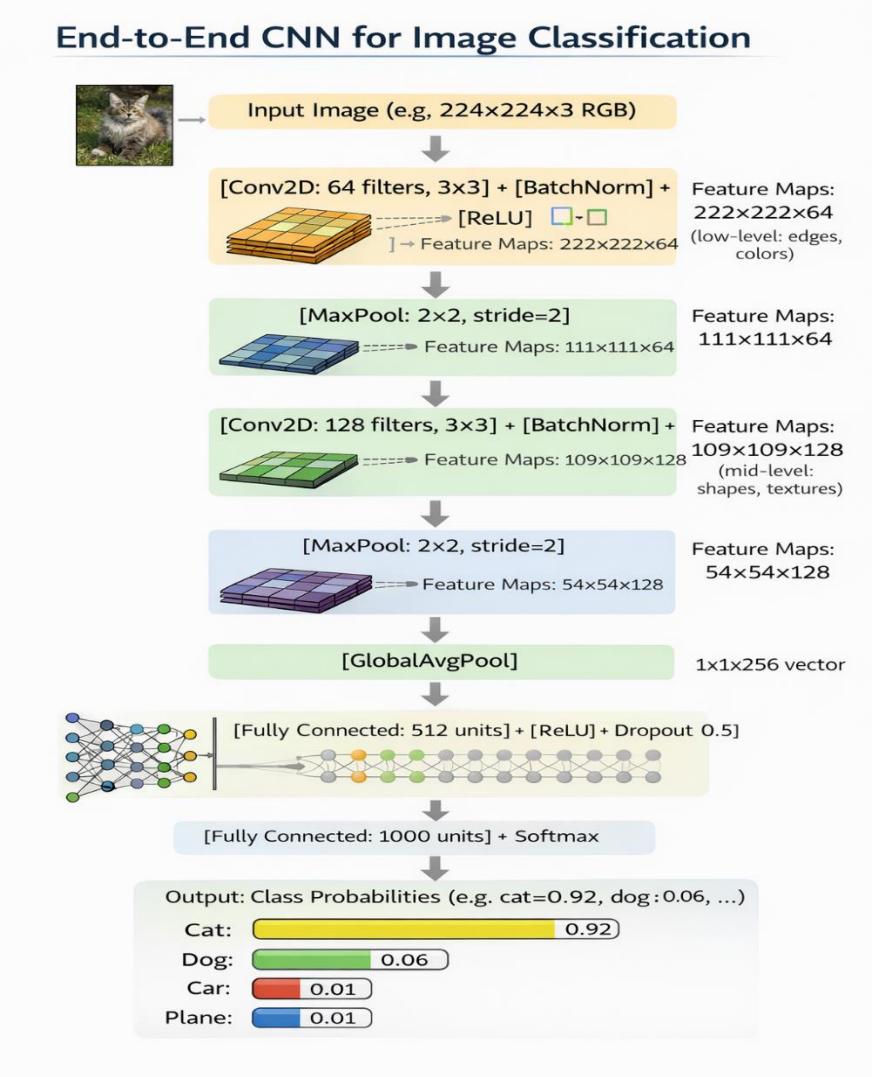
(f) Fully Connected (Dense) Layer

After the convolutional and pooling blocks, feature maps are flattened into a 1D vector and fed into fully connected layers for high-level reasoning and final classification.

(g) Softmax Output Layer

The final layer uses Softmax activation to convert raw scores (logits) into class probabilities that sum to 1, enabling multi-class classification.

3. Complete CNN Architecture Pipeline



4. Parameter Sharing and Local Connectivity

- **Parameter Sharing:** The same filter weights are applied at every position in the input. Drastically reduces parameters (a 3x3 filter has only 9 weights regardless of input size).
- **Local Connectivity:** Each neuron connects only to a small local region (receptive field), not the entire input. Captures local spatial patterns.

5. Famous CNN Architectures

Architecture	Year	Innovation	Depth
LeNet-5	1998	First practical CNN; digit recognition on MNIST	7 layers
AlexNet	2012	ReLU activation, Dropout, GPU training; won ImageNet	8 layers

VGGNet	2014	Uniform 3x3 filters throughout; very deep networks	16/19 layers
GoogLeNet	2014	Inception modules — parallel multi-scale convolutions	22 layers
ResNet	2015	Skip/residual connections — enables 100+ layer training	50-152 layers
DenseNet	2017	Every layer connects to all subsequent layers	Dense blocks
EfficientNet	2019	Compound scaling of width, depth, resolution	B0-B7 variants

6. Residual Connections (ResNet)

The key innovation of ResNet is the skip connection (shortcut). Instead of learning the desired mapping H(x), the layers learn the residual F(x) = H(x) - x. This solves the vanishing gradient problem in very deep networks.

```
Residual Block: Output = F(x, {Wi}) + x
F(x) = learned residual function (2-3 conv layers)
x    = identity shortcut (bypasses conv layers)
If gradient ∂L/∂x becomes small, the shortcut ensures
at minimum ∂L/∂x = 1 flows through — no vanishing gradient!
```

Q5. Explain Recurrent Neural Networks (RNN) in detail. What is the vanishing gradient problem? Describe LSTM and GRU as solutions. (16 Marks)

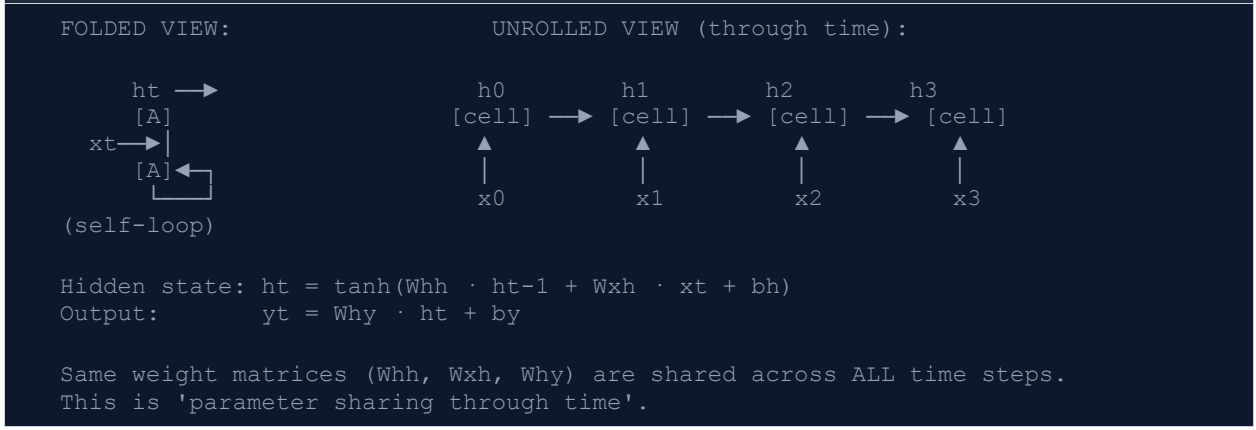
Answer:

1. Introduction to RNN

A Recurrent Neural Network (RNN) is a class of neural network designed specifically to handle sequential data — data where the order of elements carries meaningful information. Unlike feedforward networks that process inputs independently, RNNs maintain a hidden state (internal memory) that captures information from previous time steps, making them suitable for tasks such as language modeling, speech recognition, and time-series forecasting.

2. RNN Architecture and Hidden State

[DIAGRAM] RNN Unrolled Through Time




```
Hidden state: ht = tanh(Whh · ht-1 + Wxh · xt + bh)
Output:       yt = Why · ht + by

Same weight matrices (Whh, Wxh, Why) are shared across ALL time steps.
This is 'parameter sharing through time'.
```

3. Types of RNN Configurations

Configuration	Input → Output	Example Application
One-to-One	Single → Single	Standard classification (not truly recurrent)
One-to-Many	Single → Sequence	Image Captioning: one image → sentence
Many-to-One	Sequence → Single	Sentiment Analysis: sentence → positive/negative
Many-to-Many (same)	Sequence → Sequence	Video frame labeling
Many-to-Many (diff)	Sequence → Sequence	Machine Translation: English → Tamil

4. Backpropagation Through Time (BPTT)

Training RNNs requires Backpropagation Through Time (BPTT), where gradients are computed by unrolling the network across all time steps and applying standard backpropagation. This involves multiplying gradients at each time step.

```
Gradient at time step t-k:
∂L/∂Wh = Σt (∂Lt/∂ht) · ∏k (∂ht / ∂ht-k)

The product ∏(∂ht/∂ht-k) = ∏ tanh'(zt) · Whh
This is raised to the power of k (number of time steps back).
```

5. Vanishing and Exploding Gradient Problem

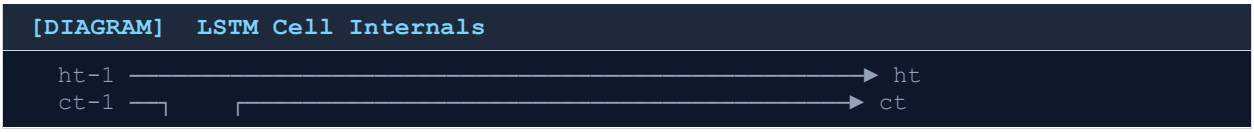
Vanishing Gradient Problem

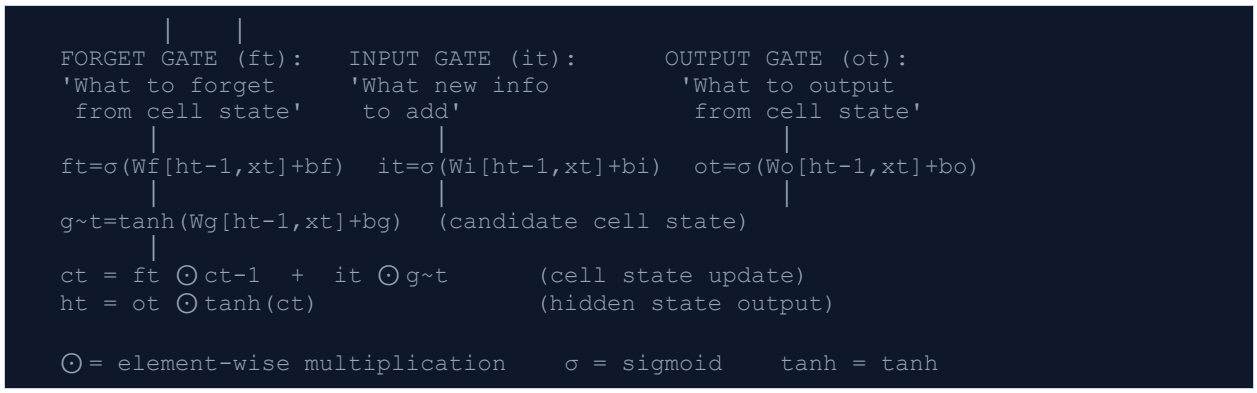
If |Whh| < 1 OR tanh'(z) is small (near saturation):
The product ∏(∂ht/∂ht-k) → 0 exponentially as k increases.
Gradients vanish → early time steps receive near-zero gradient updates.
Network fails to learn LONG-RANGE DEPENDENCIES.
e.g., 'The cat that ate the fish .... was full.' — RNN forgets 'cat' by 'was'.

If |Whh| > 1:
The product ∏(∂ht/∂ht-k) → ∞ → gradients EXPLODE → NaN / training instability.
Solution for exploding: Gradient Clipping (clip if ||g|| > threshold).
Solution for vanishing: LSTM and GRU architectures.

6. Long Short-Term Memory (LSTM)

LSTM (Hochreiter & Schmidhuber, 1997) resolves the vanishing gradient problem using a cell state (a long-term memory highway) and three gating mechanisms that control information flow.

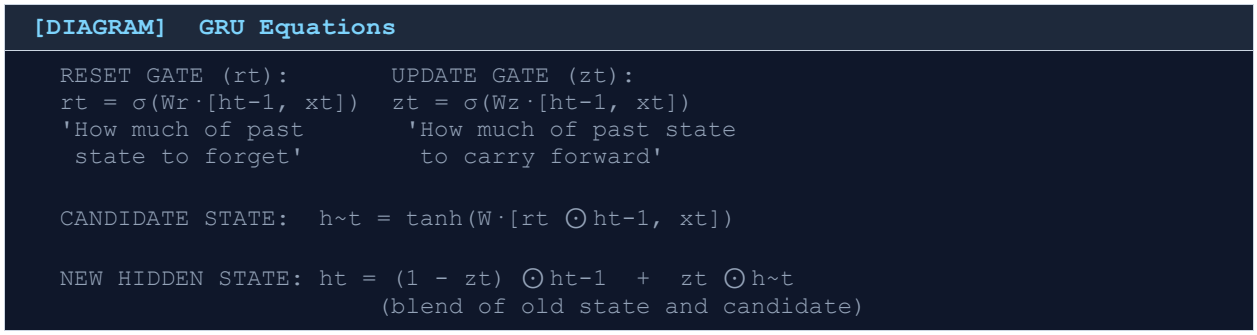




The cell state ct acts as a conveyor belt, allowing gradients to flow unchanged through time (gradient highway). The forget gate can set ft=1 to preserve cell state perfectly, preventing gradient vanishing.

7. Gated Recurrent Unit (GRU)

GRU (Cho et al., 2014) is a simplified version of LSTM with only two gates, fewer parameters, and similar performance on many tasks.



8. LSTM vs GRU Comparison

Feature	LSTM	GRU
Number of gates	3 (Forget, Input, Output)	2 (Reset, Update)
Cell state	Separate cell state + hidden state	Single hidden state
Parameters	More parameters	Fewer parameters (faster)
Long sequences	Better for very long-range deps.	Comparable for most tasks
Training speed	Slower due to more params	Faster to train
Best for	Complex sequences, NLP	Simpler sequences, smaller data

Q6. What is Feature Extraction in Deep Learning? Explain hierarchical feature learning, transfer learning, and common feature extraction techniques. (16 Marks)

Answer:

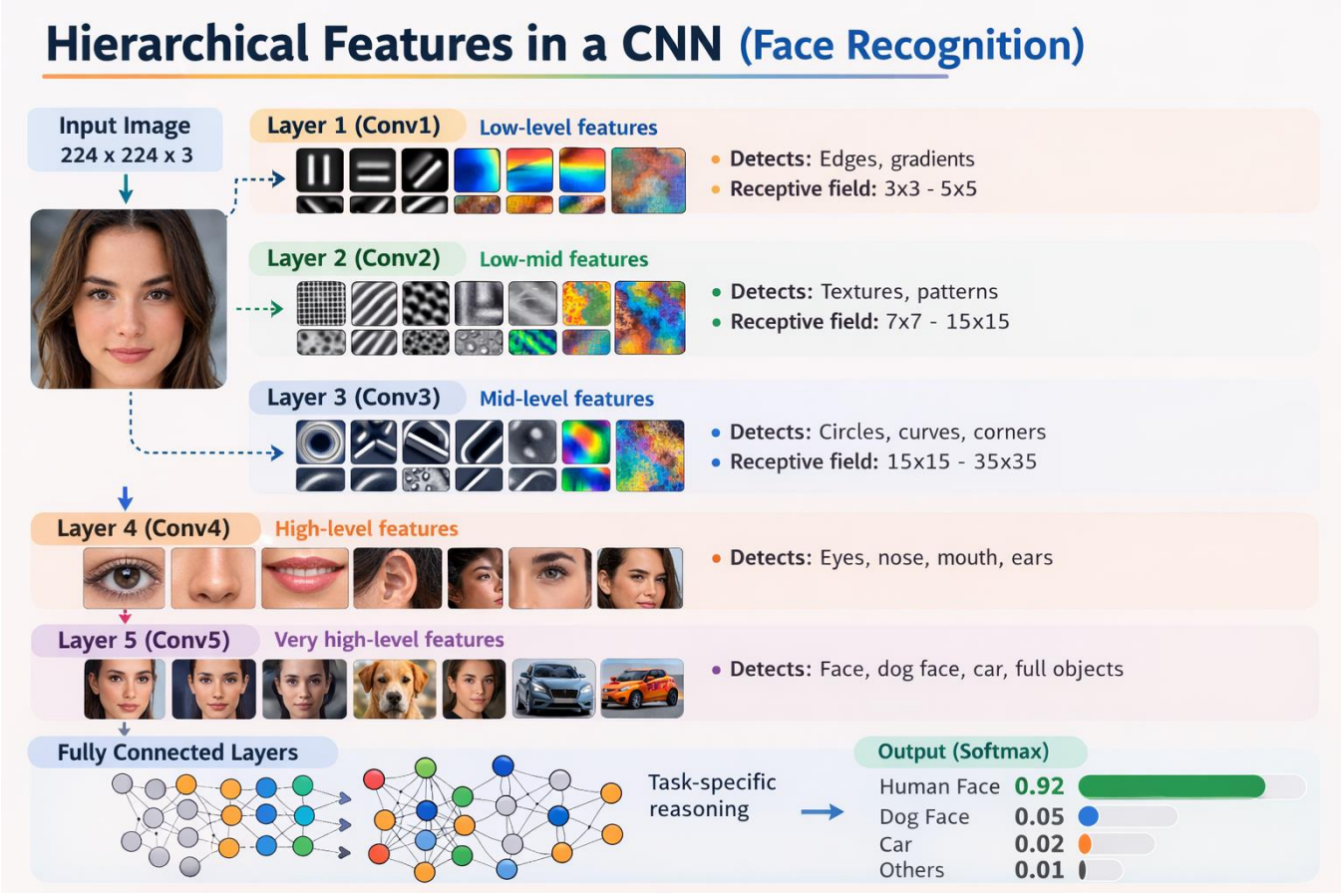
1. What is Feature Extraction?

Feature extraction is the process of transforming raw input data (pixels, audio samples, text tokens) into a numerical representation — called a feature vector — that captures the meaningful information required for a specific task. In classical machine learning, this step was performed manually by domain experts. In deep learning, the network automatically learns to extract features at multiple levels of abstraction through training.

Feature Vector	A numerical representation of an input sample that captures its important characteristics in a format suitable for downstream tasks like classification or clustering.

2. Hierarchical Feature Learning in CNNs

One of the most powerful properties of deep networks is hierarchical representation learning — each layer builds on the representations of the previous layer to capture increasingly abstract features.



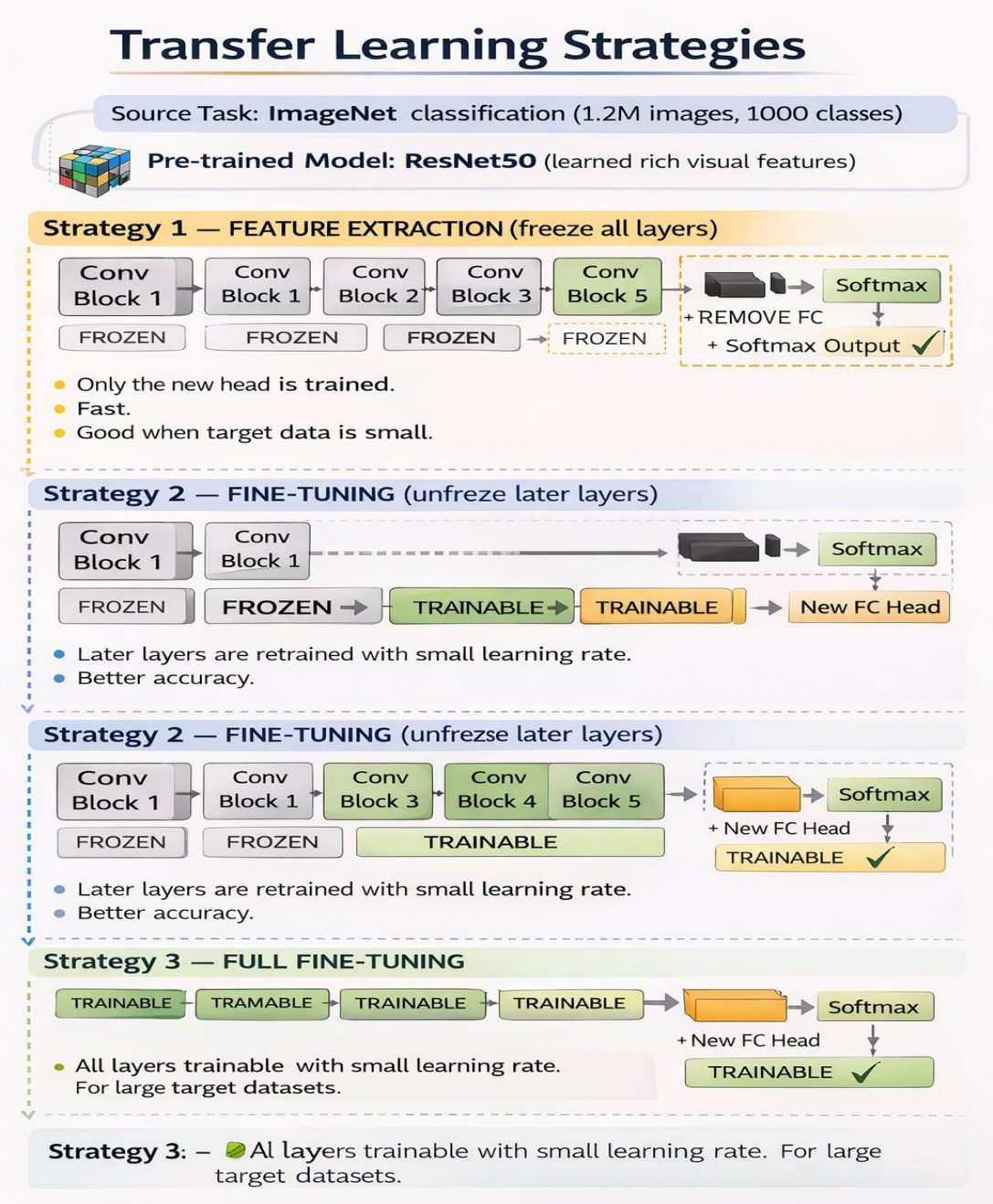
3. Why Hierarchical Features Work

- Lower layers learn general, reusable features (applicable to any image task).
- Higher layers learn task-specific, abstract features.
- This mirrors how the human visual cortex processes information progressively.

- The learned hierarchy makes CNNs far more efficient than hand-crafted features.

4. Transfer Learning

Transfer learning is a technique where a model trained on one large task (source domain) is reused as the starting point for a different but related task (target domain). This is possible because lower-layer features are general and transferable.



5. Benefits of Transfer Learning

11. Requires significantly less labeled training data.
12. Much faster training (fewer epochs needed).
13. Achieves higher accuracy even with small datasets.
14. Reduces the risk of overfitting on small datasets.

6. Common Feature Extraction Techniques in DL

Technique	Description & Application
CNN Feature Maps	Intermediate conv layer outputs capture spatial feature patterns; used for visualization and retrieval
Global Average Pooling (GAP)	Averages each feature map to a single value; replaces FC layers, reduces overfitting
Word Embeddings	Represent words as dense vectors (Word2Vec, GloVe, BERT); captures semantic relationships
Autoencoders	Encoder compresses input to latent space; decoder reconstructs. Latent vector = features
Variational Autoencoders	Learns continuous latent distributions; enables generative sampling
Siamese Networks	Learn to compare two inputs; used in face verification, signature matching

7. Embedding Spaces

Embeddings are dense, low-dimensional feature representations learned by the network. In NLP, word2vec learns that 'king' - 'man' + 'woman' ≈ 'queen', showing meaningful arithmetic in feature space. In vision, similar-looking images cluster together in the embedding space, enabling image retrieval and few-shot learning.

Q7. Explain Deep Belief Networks (DBN) in detail — architecture, training procedure, and applications. (16 Marks)

Answer:

1. Introduction to Deep Belief Networks

A Deep Belief Network (DBN) is a probabilistic generative model composed of multiple layers of Restricted Boltzmann Machines (RBMs) stacked on top of each other. Introduced by Geoffrey Hinton, Simon Osindero, and Yee-Whye Teh in 2006, DBNs were one of the first deep architectures to be successfully trained, breaking the barrier that had prevented researchers from effectively training deep neural networks for nearly two decades.

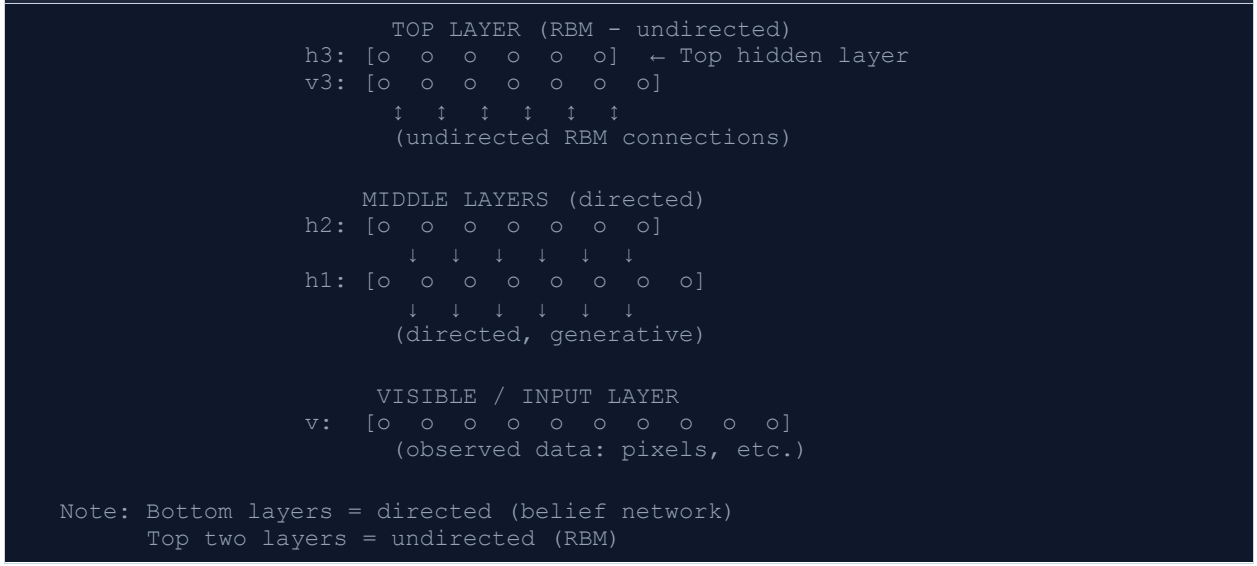
DBN	A generative graphical model: multiple layers of RBMs where each layer serves as the visible layer for the next, enabling deep hierarchical representation learning through greedy layer-wise pretraining.

2. Historical Significance

Before 2006, training deep networks was practically impossible due to the vanishing gradient problem and poor weight initialization. Hinton's insight was to use Restricted Boltzmann Machines to pretrain each layer one at a time (greedy layer-wise pretraining). This provided a good weight initialization for the entire deep network, after which the whole network could be fine-tuned with backpropagation. This breakthrough reignited interest in deep learning and triggered the modern DL revolution.

3. DBN Architecture

[DIAGRAM] Deep Belief Network Structure



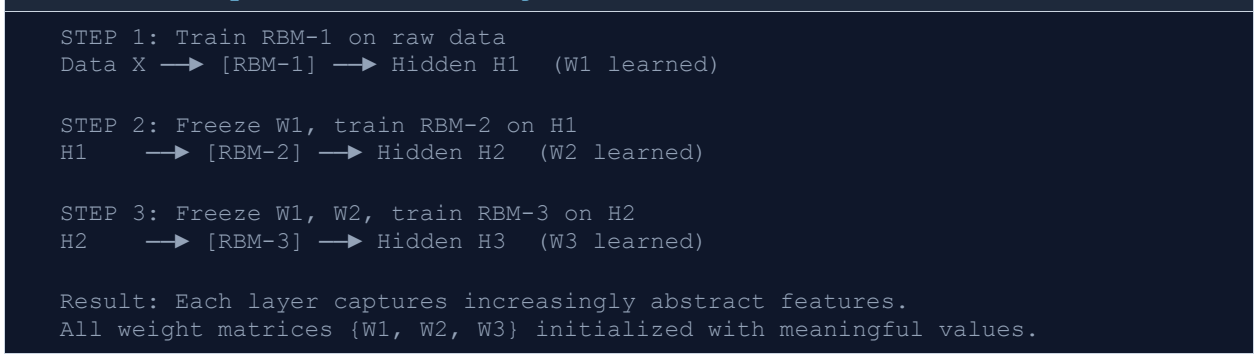
4. DBN Training Procedure

DBN training proceeds in two distinct phases:

Phase 1: Greedy Layer-Wise Unsupervised Pretraining

15. Train the first RBM (Layer 1) using the raw input data as visible units. Learn weights W1 that capture statistical regularities in the input. Use Contrastive Divergence (CD) algorithm.
16. Fix the learned weights W1 of the first RBM. Run the input data through it to obtain the hidden representations h1. These activations become the 'data' for the second RBM.
17. Train the second RBM (Layer 2) using h1 as visible input. Learn weights W2 that capture higher-level features.
18. Repeat for each additional layer: each layer learns to model the hidden representations of the layer below it.

[DIAGRAM] Layer-Wise Pretraining Procedure



Phase 2: Supervised Fine-Tuning

19. After pretraining, add a softmax classification layer on top.
20. Unfold the entire network (use pretrained weights as initialization).
21. Fine-tune all weights simultaneously using standard backpropagation and labeled training data.

22. The pretrained weights provide a much better starting point than random initialization, leading to faster convergence and better final accuracy.

5. Why Greedy Pretraining Works

Theoretical Justification (Hinton et al., 2006)
Each time a new RBM layer is added and trained on top of the previous hidden representations, it can be shown that the overall variational lower bound on the log-likelihood of the data is improved. In other words, each additional trained layer always makes the model a better generative model of the data — guaranteed improvement with each added layer.

6. DBN as Generative Model

Beyond discriminative (classification) tasks, DBNs can also generate new samples by performing top-down ancestral sampling: sample from the top RBM, then pass the sample down through the directed layers to generate a new data point (e.g., a new digit image). This makes DBNs valuable as generative models.

7. Applications of DBN

Application Domain	Specific Use Case
Image Recognition	Feature extraction, digit/handwriting recognition
Speech Processing	Acoustic modeling for speech recognition (replaces Gaussian Mixture Models)
Natural Language Processing	Document classification, sentiment analysis, topic modeling
Collaborative Filtering	Recommendation systems (Netflix movie recommendations)
Drug Discovery	Predicting molecular properties, virtual screening of compounds
Anomaly Detection	Learning normal data distribution; flagging deviations as anomalies

Q8. Describe Restricted Boltzmann Machines (RBM) in detail — structure, energy function, probability model, and Contrastive Divergence training algorithm. (16 Marks)

Answer:

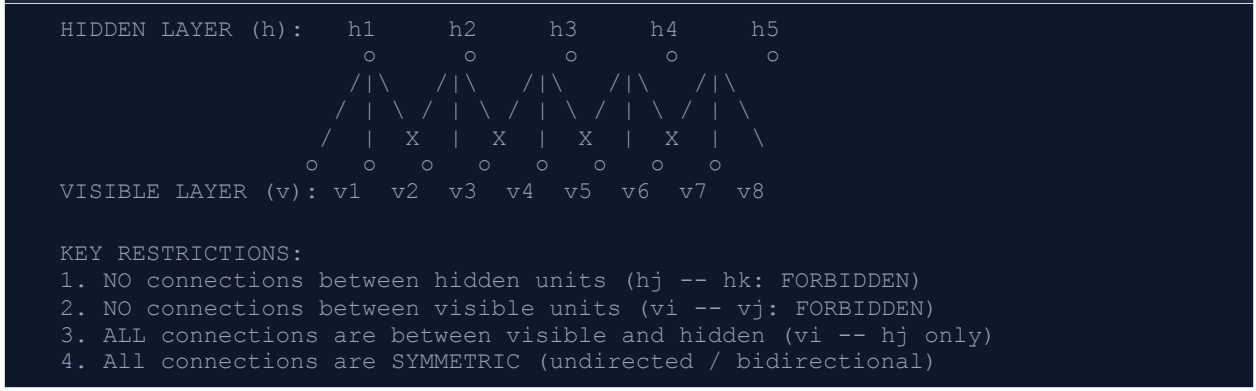
1. Introduction to Boltzmann Machines

A Boltzmann Machine is a type of stochastic recurrent neural network that can learn a probability distribution over its inputs. It is an energy-based model inspired by statistical mechanics — the network seeks configurations of minimum energy, and the probability of a configuration is determined by its energy. However, the fully connected Boltzmann Machine is computationally intractable for practical use.

2. Restricted Boltzmann Machine (RBM)

An RBM is a simplified Boltzmann Machine with a crucial restriction: there are no connections between units within the same layer. This creates a bipartite graph structure with only connections between the visible and hidden layers, making inference and training tractable.

[DIAGRAM] RBM Bipartite Graph Structure



3. Energy Function

Like all Boltzmann Machines, the RBM is defined through an energy function. The probability of a joint configuration (v, h) is determined by its energy — lower energy configurations are more probable.

```
Energy Function:
E(v, h) = - Σi ai·vi  -  Σj bj·hj  -  Σij vi·wij·hj

Where:
  vi = visible unit i (binary: 0 or 1)
  hj = hidden unit j (binary: 0 or 1)
  ai = bias of visible unit i
  bj = bias of hidden unit j
  wij = weight connecting visible unit i to hidden unit j

Joint Probability:
P(v, h) = (1/Z) · exp(-E(v, h))

Z = Σv Σh exp(-E(v,h))  ← partition function (normalizing constant)
Z is intractable to compute exactly (exponential sum)
```

4. Conditional Probabilities

Due to the bipartite structure (no intra-layer connections), the hidden units are conditionally independent given the visible units, and vice versa. This makes inference tractable — a major advantage over the general Boltzmann Machine.

```
P(hj = 1 | v) = σ( bj + Σi vi · wij )   ← hidden given visible
P(vi = 1 | h) = σ( ai + Σj hj · wij )   ← visible given hidden

σ(x) = 1 / (1 + e^(-x))  (sigmoid function)

This allows efficient block Gibbs sampling:
Sample ALL hidden units simultaneously given visible (one step)
```

```
Sample ALL visible units simultaneously given hidden (one step)
```

5. Contrastive Divergence (CD) Training

The goal of training is maximum likelihood — find weights that maximize P(v) = Σh P(v,h) for the training data. Direct gradient computation requires summing over all 2^(n+m) configurations (intractable). Contrastive Divergence (CD-k), proposed by Hinton (2002), provides an efficient approximation.

[DIAGRAM] Contrastive Divergence (CD-1) Algorithm

```
FOR each training sample v0:

=== POSITIVE PHASE (data-driven statistics) ===
Step 1: Clamp visible units to training data: v = v0
Step 2: Sample hidden:  h0 ~ P(h | v0)
        P(hj=1|v0) = σ(bj + Σi v0i · wij)
Compute positive statistics: <vi · hj>data = v0i · h0j

=== NEGATIVE PHASE (model-driven statistics) ===
Step 3: Reconstruct visible: v1 ~ P(v | h0)
        P(vi=1|h0) = σ(ai + Σj h0j · wij)
Step 4: Re-sample hidden:   h1 ~ P(h | v1)
        P(hj=1|v1) = σ(bj + Σi v1i · wij)
Compute negative statistics: <vi · hj>model = v1i · h1j

=== UPDATE WEIGHTS ===
Δwij = η · ( v0i·h0j  -  v1i·h1j )   (positive - negative phase)
Δai  = η · ( v0i  -  v1i )            (visible bias update)
Δbj  = η · ( h0j  -  h1j )            (hidden bias update)

Intuition: Increase weights for patterns in the DATA,
           Decrease weights for patterns the MODEL hallucinates.
```

6. CD-k: Multiple Gibbs Steps

CD-1 uses only 1 step of Gibbs sampling for the negative phase, which is a rough approximation but works well in practice. CD-k performs k Gibbs steps before sampling the negative statistics — this gives a better estimate of the model distribution at the cost of more computation. CD-1 is most commonly used.

7. RBM as Feature Extractor

After training, the hidden activations P(h|v) for any input v serve as a feature representation. The RBM has learned to transform the input into a distributed code in the hidden layer that captures statistical structure. These learned features form the building blocks of Deep Belief Networks when RBMs are stacked.

8. Boltzmann Machine vs RBM — Summary

Aspect	Full Boltzmann Machine	Restricted Boltzmann Machine
Connections	Intra-layer + inter-layer	Inter-layer ONLY
Inference	Intractable (MCMC required)	Tractable (closed-form conditionals)
Training	Very slow, impractical	Efficient (Contrastive Divergence)

Hidden independence	No	Yes (given visible)
Practical use	Rarely used directly	Building block of DBNs

Q9. Explain the challenges in training Deep Neural Networks and the techniques used to overcome them — including regularization, initialization, batch normalization, and optimization. (16 Marks)

Answer:

1. Challenges in Training Deep Neural Networks

- Vanishing / Exploding Gradients: Gradients become too small or too large in deep networks during backpropagation.
- Overfitting: Model memorizes training data but fails to generalize to unseen data.
- Slow Convergence: Training takes too many epochs to converge; poor local minima.
- Covariate Shift: Distribution of inputs to each layer changes as weights update (internal covariate shift).
- Hyperparameter Sensitivity: Network performance is highly sensitive to learning rate, architecture choices.
- Computational Cost: Deep networks require large memory and GPU time.

2. Weight Initialization

Proper initialization of weights is critical. All-zero initialization fails (symmetry breaking problem — all neurons learn the same thing). Large random initialization causes vanishing/exploding gradients from the start.

Method	Formula / Strategy	Best For
Zero Init	All w = 0	NEVER use — symmetry problem
Random Normal	w ~ N(0, 0.01)	Shallow networks only
Xavier / Glorot	w ~ N(0, 2/(nin+nout))	Sigmoid / Tanh activations
He (Kaiming)	w ~ N(0, 2/nin)	ReLU / Leaky ReLU activations
Pretrained	Transfer from trained model	Transfer learning scenarios

3. Batch Normalization (BatchNorm)

Batch Normalization (loffe & Szegedy, 2015) normalizes the inputs to each layer to have zero mean and unit variance, computed over the current mini-batch. This addresses the internal covariate shift problem and significantly stabilizes and accelerates training.

```
  For a mini-batch B = {x1, x2, ..., xm}:
  μB = (1/m) Σ xi                (batch mean)
  σ²B = (1/m) Σ (xi - μB)²       (batch variance)
  x^i = (xi - μB) / √(σ²B + ε)   (normalize, ε for numerical stability)
  yi = γ · x^i + β               (scale and shift: γ, β are learned
parameters)
```

```
γ, β allow the network to learn the optimal scale and shift,
so BatchNorm does not forcibly constrain the representation.
```

Benefits of Batch Normalization:

23. Allows much higher learning rates, speeding up training.
24. Reduces sensitivity to weight initialization.
25. Acts as a regularizer, reducing the need for Dropout in some cases.
26. Reduces the vanishing gradient problem by keeping activations in a healthy range.

4. Regularization Techniques

(a) L2 Regularization (Weight Decay)

```
L_total = L_loss + λ · Σ wi²
Gradient update: w ← w - η·(∂L/∂w + 2λ·w) = w·(1 - 2ηλ) - η·∂L/∂w
Effect: Shrinks all weights towards zero proportionally.
        Prevents any single weight from becoming too dominant.
```

(b) L1 Regularization

```
L_total = L_loss + λ · Σ |wi|
Effect: Drives some weights exactly to zero → sparse networks.
        Acts as automatic feature selection.
```

(c) Dropout

During each training forward pass, each neuron is independently set to zero with probability p (the dropout rate). At test time, all neurons are active but their outputs are scaled by (1-p). This forces the network to learn multiple redundant representations, acting as an ensemble of many different neural networks.

[DIAGRAM] Dropout Mechanism

```
TRAINING:                          TESTING:
[x1] [x2] [x3] [x4] [x5]          [x1] [x2] [x3] [x4] [x5]
  ↓    X    ↓    X    ↓             ↓    ↓    ↓    ↓    ↓
[h1] [--] [h3] [--] [h5]          [h1/p][h2/p][h3/p][h4/p][h5/p]
(dropped neurons shown as --)     (scale by 1/p or equivalently
                                   multiply weights by p at train time)
```

(d) Early Stopping

Monitor validation loss after each epoch. Stop training when validation loss stops decreasing (begins to increase), even if training loss continues decreasing. Save the model weights at the point of minimum validation loss.

(e) Data Augmentation

Artificially expand the training dataset by applying random transformations: horizontal flipping, rotation, cropping, brightness/contrast changes, cutout, mixup. Forces the model to be invariant to these transformations.

5. Optimization Algorithms

[DIAGRAM] Learning Rate and Optimization Landscape

```
Loss Surface:     Too High LR:        Good LR:          Too Low LR:
   /\              Oscillates/         Smooth            Extremely
  /  \             diverges            convergence       slow
 /    \_____/\
(Global Min)  (Local Min)

SGD with Momentum:
v(t) = β·v(t-1) + η·∇L     (velocity accumulation)
w(t) = w(t-1) - v(t)       (weight update with momentum)

Adam Optimizer (most popular):
m(t) = β1·m(t-1) + (1-β1)·∇L       (first moment: mean)
v(t) = β2·v(t-1) + (1-β2)·(∇L)²    (second moment: variance)
m^(t) = m(t)/(1-β1^t)               (bias correction)
v^(t) = v(t)/(1-β2^t)               (bias correction)
w(t) = w(t-1) - η · m^(t)/√(v^(t)+ε)
```

6. Learning Rate Scheduling

Schedule	Description	When to Use
Step Decay	Multiply LR by factor γ every k epochs	Simple, predictable decay
Exponential Decay	LR = LR0 · e^(-λt)	Smooth continuous decay
Cosine Annealing	LR follows cosine curve to near-zero	State-of-the-art in practice
Warm Restarts	Periodically reset LR to initial value	Escaping local minima
Warmup	Start with low LR, ramp up, then decay	Transformer training

7. Gradient Clipping (for Exploding Gradients)

```
If ||g|| > threshold:
    g ← g · (threshold / ||g||)     (scale gradient to max norm)
Commonly used in RNN/LSTM training where gradients can explode.
```

Q10. Discuss in detail the applications of Deep Learning across various domains. Include specific architectures used in each domain. (16 Marks)

Answer:

1. Introduction

Deep Learning has transformed numerous industries by enabling machines to perform tasks that were previously believed to require human intelligence. The ability to learn complex hierarchical representations from raw data makes DL applicable across virtually every domain that involves large amounts of data and complex pattern recognition.

2. Computer Vision

(a) Image Classification

Image classification assigns a label to an entire image. Deep CNNs (ResNet, VGG, EfficientNet) trained on ImageNet achieve superhuman accuracy on this task. Applications include medical diagnosis (classifying X-ray/MRI scans), quality control in manufacturing, and content categorization on social media platforms.

(b) Object Detection

Object detection identifies and localizes multiple objects in an image by predicting bounding boxes and class labels. Key architectures:

- **YOLO (You Only Look Once):** Real-time detector, processes entire image in one forward pass; used in autonomous vehicles and surveillance.
- **Faster R-CNN:** Two-stage detector with a Region Proposal Network; high accuracy; used in medical imaging.
- **SSD (Single Shot MultiBox Detector):** Balance of speed and accuracy for mobile applications.

(c) Image Segmentation

Segmentation assigns a class label to every pixel in an image (semantic segmentation) or identifies individual object instances (instance segmentation).

- **U-Net:** Encoder-decoder architecture with skip connections; widely used in medical image segmentation (tumor detection, organ segmentation).
- **DeepLab / Mask R-CNN:** State-of-the-art for semantic and instance segmentation in autonomous driving.

(d) Face Recognition

- **DeepFace (Facebook) / FaceNet (Google):** Deep CNNs trained with metric learning achieve 99%+ accuracy on face verification benchmarks.

3. Natural Language Processing (NLP)

(a) Language Modeling and Text Generation

Large Language Models (LLMs) like GPT-3 and GPT-4 are trained on billions of text tokens to predict the next word. They can generate coherent, contextually appropriate text for a wide range of tasks including writing assistance, code generation, and question answering. ChatGPT is a prominent example built on this technology.

(b) Machine Translation

The Transformer architecture with self-attention (Vaswani et al., 2017) revolutionized machine translation. Google Translate, DeepL, and other services use deep Transformer models to translate between 100+ language pairs with near-human quality.

(c) Sentiment Analysis and Text Classification

BERT (Bidirectional Encoder Representations from Transformers) achieves state-of-the-art results on text classification, sentiment analysis, and named entity recognition by leveraging bidirectional context understanding.

(d) Question Answering and Information Retrieval

DL models can read a passage and answer specific questions about it (extractive QA). Systems like BERT and GPT-4 are used in enterprise knowledge management, search engines, and customer support automation.

4. Speech and Audio Processing

Task	Architecture / System	Details
Speech Recognition (ASR)	DeepSpeech, Whisper (OpenAI), Wav2Vec	RNN/Transformer-based models; real-time transcription
Text-to-Speech (TTS)	WaveNet (DeepMind), Tacotron 2	CNN/RNN architectures; human-like synthetic speech
Speaker Identification	d-vectors, x-vectors (deep embeddings)	Verify or identify speaker identity
Music Generation	MuseNet, Jukebox (OpenAI)	Generate music in various styles and genres
Keyword Spotting	Tiny neural nets on mobile	Voice activation ('Hey Siri', 'OK Google')

5. Healthcare and Biomedical Applications

(a) Medical Image Analysis

CNNs analyze medical images with accuracy rivaling or exceeding specialist physicians. Applications include detection of diabetic retinopathy from fundus photographs, skin cancer classification from dermoscopy images, chest X-ray analysis for pneumonia and COVID-19, and histopathology slide analysis for cancer grading.

(b) Drug Discovery

Deep learning accelerates drug discovery by predicting molecular properties, identifying drug-protein binding affinities, and generating novel drug candidates using generative models (VAEs, Generative Adversarial Networks).

(c) Protein Structure Prediction

AlphaFold 2 (DeepMind, 2020) uses a deep transformer-based architecture to predict protein 3D structures from amino acid sequences with atomic-level accuracy. This breakthrough, called 'the biggest problem in biology solved', has transformed structural biology and drug design.

6. Autonomous Vehicles

- Perception: CNNs for real-time object detection (pedestrians, vehicles, signs, lane markings).
- Depth Estimation: Monocular depth estimation from a single camera using CNNs.
- Sensor Fusion: Combining data from cameras, LiDAR, and radar using deep learning.
- Path Planning: Reinforcement learning for decision making and navigation.
- Examples: Tesla Autopilot, Waymo, NVIDIA DRIVE platform.

7. Robotics and Reinforcement Learning

- Robot grasping: CNNs estimate grasp pose and success probability from RGB-D images.
- Locomotion: Deep RL (PPO, SAC) enables robots to learn walking, running, and complex maneuvers.
- Game Playing: AlphaGo, AlphaZero use deep RL with MCTS to master Chess, Go, and Shogi at superhuman level. OpenAI Five defeated professional Dota 2 players.

8. Generative AI

Generative Adversarial Networks (GANs)

GANs (Goodfellow, 2014) consist of a Generator (creates fake data) and a Discriminator (distinguishes real from fake) trained adversarially. Applications: photorealistic image synthesis (StyleGAN), image-to-image translation (pix2pix), deepfakes, and art generation.

Diffusion Models

Diffusion models (DDPM, Stable Diffusion, DALL-E) learn to reverse a noise-adding process to generate high-quality images from text prompts. They currently represent the state of the art in image generation quality.

9. Finance and Business

Application	Description
Fraud Detection	LSTM/CNN models analyze transaction sequences for anomalous patterns
Algorithmic Trading	RL agents and LSTM models predict price movements and execute trades
Credit Scoring	Deep learning models assess creditworthiness from diverse data sources
Recommendation Systems	Netflix, YouTube, Spotify use deep collaborative filtering (NCF, DL4Rec)
Document Processing	Document classification, information extraction from contracts/forms

Summary: Deep Learning Impact

Deep Learning has achieved superhuman performance in:

- Image recognition (ImageNet 2015 — ResNet surpassed human-level accuracy)
- Speech recognition (Word error rate below human-level transcription)
- Game playing (Go, Chess, Atari — superhuman performance)
- Protein structure prediction (AlphaFold — 92%+ accuracy)

These breakthroughs are enabled by deep architectures (CNN, RNN, Transformers), massive datasets, GPU computing, and advanced training techniques.