Northeastern University
Khoury College of
Computer Sciences

CNN & Unsupervised Learning

DS 4400 | Machine Learning and Data Mining I

Zohair Shafi

Spring 2026

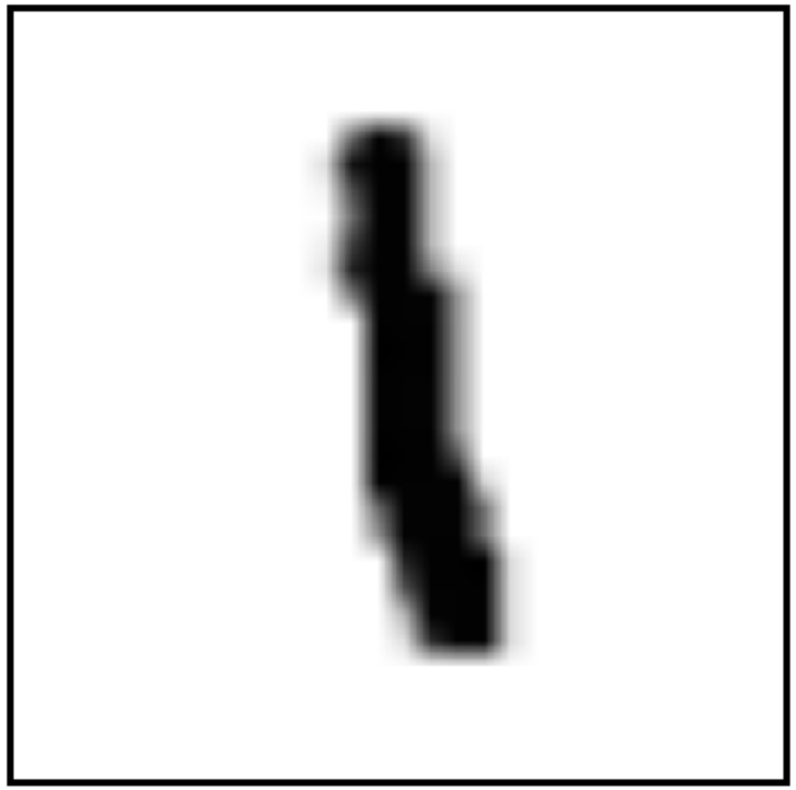
Monday | March 23rd, 2026

Today’s Outline

- Convolutional Neural Networks
- Unsupervised Learning

Convolutional Neural Networks

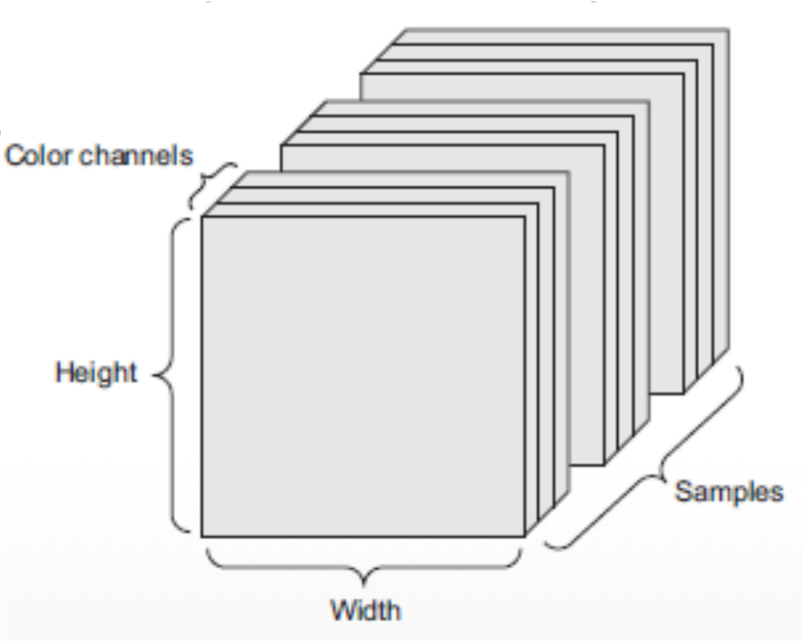
- An image is represented as a grid of pixel values



$\approx$

$$\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .6 & .8 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .5 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .7 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & .9 & 1 & .1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & .3 & 1 & .1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

Convolutional Neural Networks



- An image is represented as a grid of pixel values
- Lets say this image os 224x224 RGB image
  - Input size after flattening out image = 224 x 224 x 3 = 150,528
  - Input of neural network needs 150,528 neurons
  - Assume first hidden layer has 1000 neurons, first weight matrix will have 150,528 x 1000 =150,528,000 ~ 150 million parameters in the first layer alone.

Convolutional Neural Networks

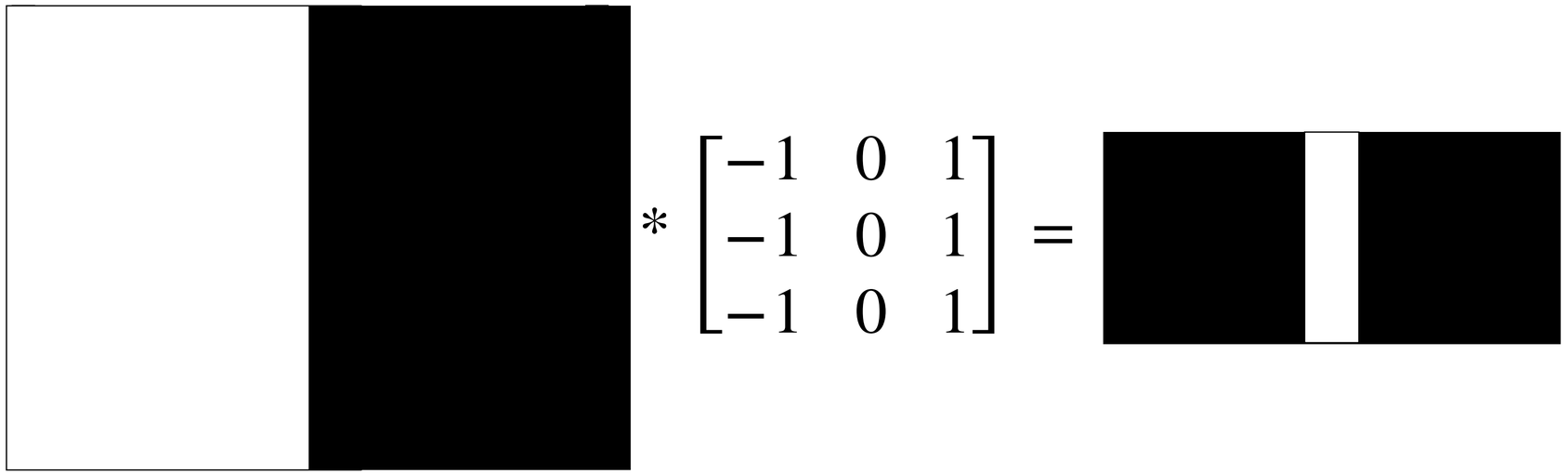
- CNNs exploit these insights through:
  - **Local receptive fields**: Each neuron sees only a small region
  - **Weight sharing:** Same detector (filter) applied across entire image
  - **Pooling:** Reduces spatial dimensions, adds invariance

Convolutional Neural Networks

- These filters “slide” over the entire dataset
- Each filter learns something about the data
- 2D filters similarly slide over an image

Convolutional Neural Networks

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$ — Vertical Edge Filter



Convolutional Neural Networks

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$ — Vertical Edge Filter

$$\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \end{bmatrix}$$

Convolutional Neural Networks

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$ — Vertical Edge Filter

$$\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \end{bmatrix}$$

Convolutional Neural Networks

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$ — Vertical Edge Filter

$$\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \end{bmatrix}$$

Convolutional Neural Networks

Fixed Filters

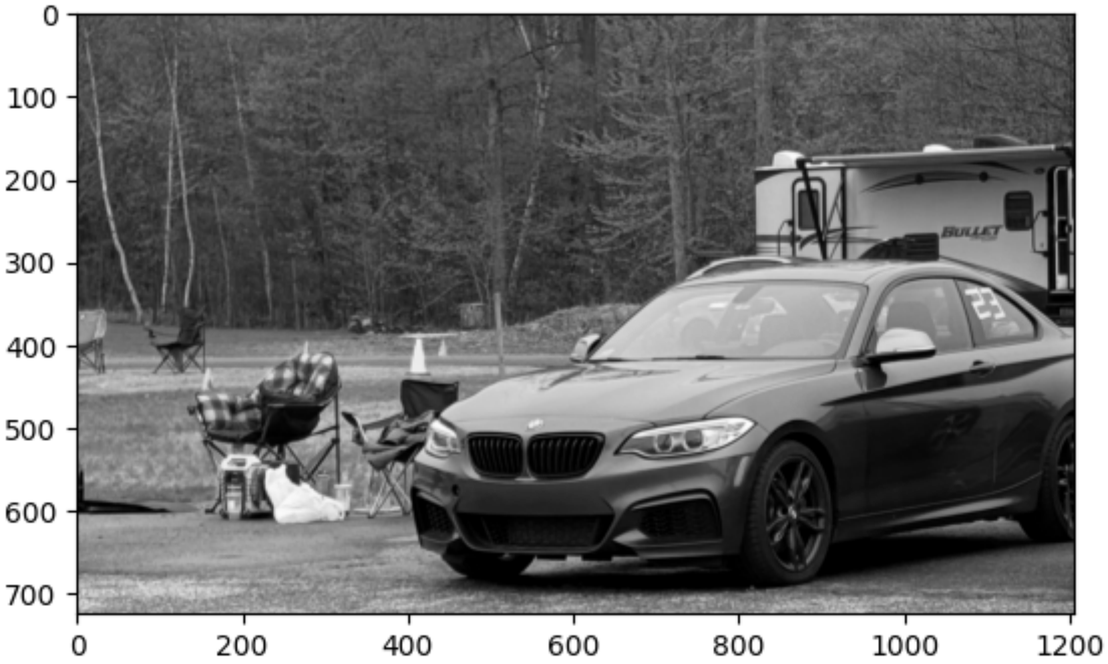
$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$ — Vertical Edge Filter

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$ — Blur (Averaging)

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$ — Horizontal Edge Filter

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$ — Sharpen

Convolutional Neural Networks

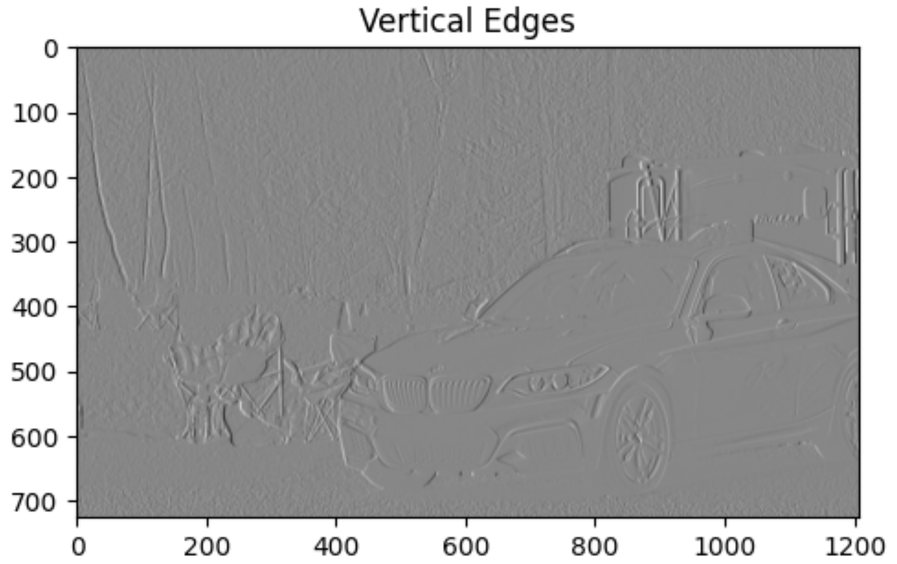


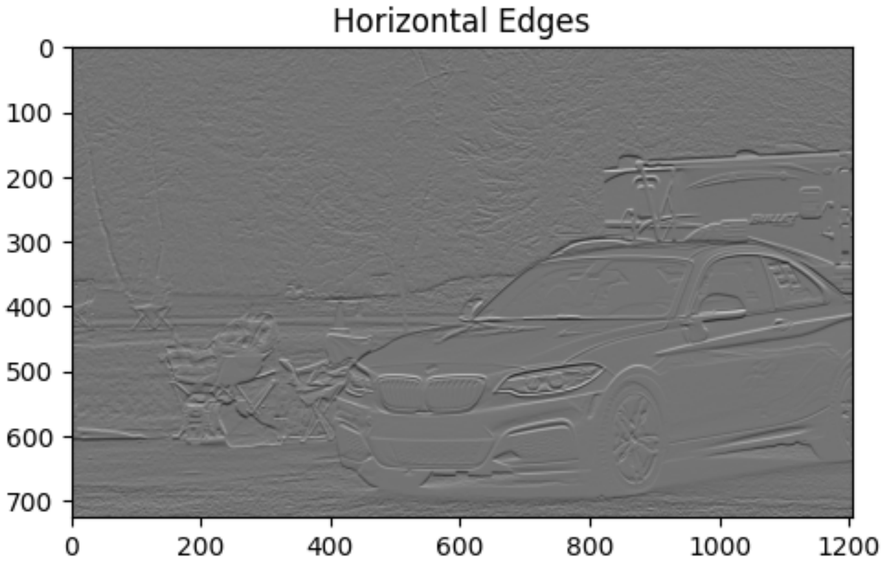
Original

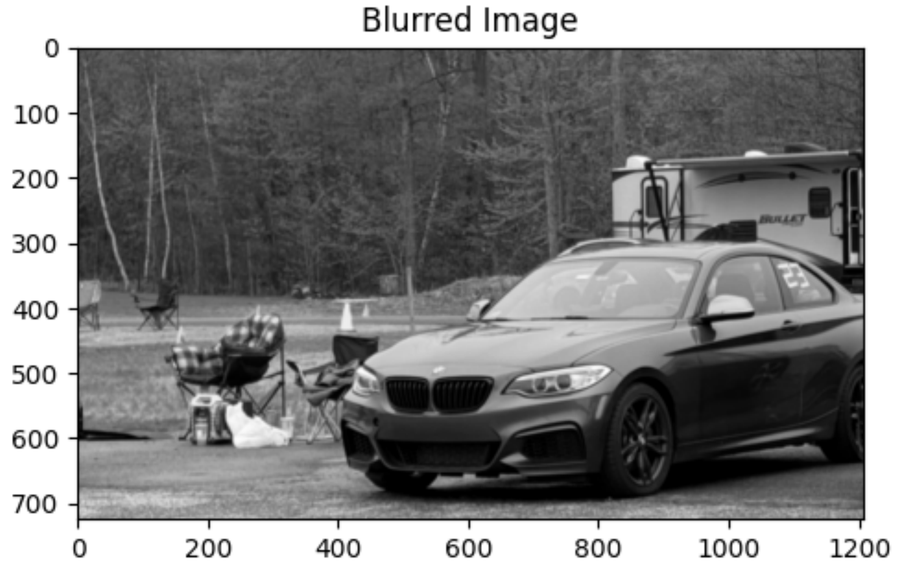
Convolutional Neural Networks

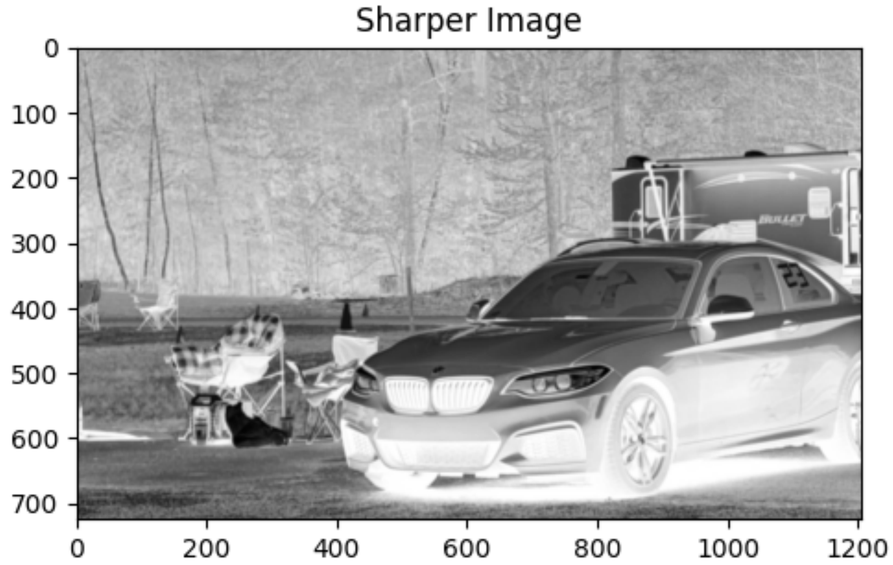
These are called activation maps

Filters of the next layer are applied on these activation maps









Convolutional Neural Networks

- Key idea of CNNs - **learn filter values** rather than hand designing them
- A convolutional layer has **multiple filters**, each producing one output channel (feature map).

Convolutional Neural Networks

- Key idea of CNNs - **learn filter values** rather than hand designing them
- A convolutional layer has **multiple filters**, each producing one output channel (feature map).
- Instead of having a fixed vertical edge filter like

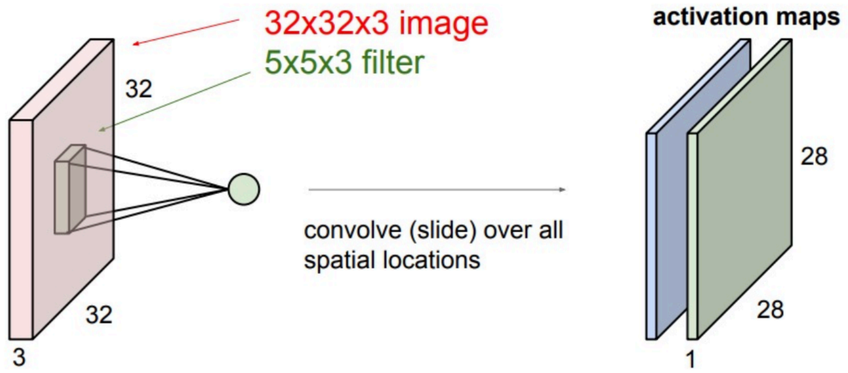
$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

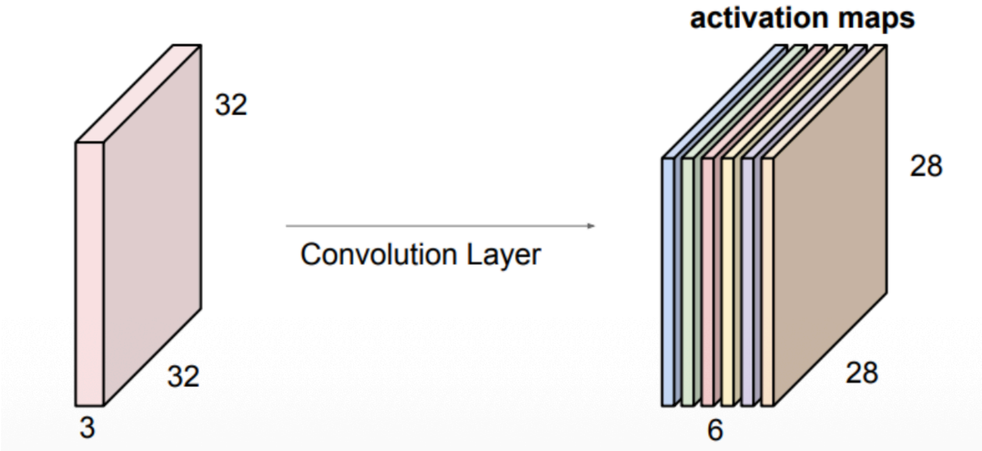
- You learn filters like

$$\text{Filter 1: } \begin{bmatrix} \theta_{10} & \theta_{11} & \theta_{12} \\ \theta_{13} & \theta_{14} & \theta_{15} \\ \theta_{16} & \theta_{17} & \theta_{18} \end{bmatrix} \quad \text{Filter 2: } \begin{bmatrix} \theta_{20} & \theta_{21} & \theta_{22} \\ \theta_{23} & \theta_{24} & \theta_{25} \\ \theta_{26} & \theta_{27} & \theta_{28} \end{bmatrix}$$

Convolutional Neural Networks

- Input: $H_{in} \times W_{in} \times C_{in}$ (height x width x channels)
- Output: $H_{out} \times W_{out} \times C_{out}$ (one channel per filter)
- Each filter has shape $k \times k \times C_{in}$
- **Example**
  - Input Image: $32 \times 32 \times 3$ (RGB Image), 6 filters
  - Each filter: $5 \times 5 \times 3 = 75$ parameters
  - Output Shape: $28 \times 28 \times 6$ (why 28?)
  - Total parameters = 75 x 6 = 450, compared to 32 x 32 x 3 x 1000 = 3M parameters for MLP





Convolutional Neural Networks

Pooling

- Pooling reduces spatial dimensions, providing:
  - Translation invariance
  - Reduced computation
  - Larger receptive field
- Example - **Max** Pool = (3x3) with stride = 3

$$\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 10 & 0 \\ 10 & 0 \end{bmatrix}$$

Convolutional Neural Networks

Pooling

- Pooling reduces spatial dimensions, providing:
  - Translation invariance
  - Reduced computation
  - Larger receptive field
- Example - **Max** Pool = (3x3) with stride = 3

$$\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 10 & 0 \\ 10 & 0 \end{bmatrix}$$

Convolutional Neural Networks

Pooling

- Pooling reduces spatial dimensions, providing:
  - Translation invariance
  - Reduced computation
  - Larger receptive field
- Example - **Max** Pool = (3x3) with stride = 3

$$\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 10 & 0 \\ 10 & 0 \end{bmatrix}$$

Convolutional Neural Networks

Pooling

- Pooling reduces spatial dimensions, providing:
  - Translation invariance
  - Reduced computation
  - Larger receptive field
- Example - **Max** Pool = (3x3) with stride = 3

$$\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 10 & 0 \\ 10 & 0 \end{bmatrix}$$

Convolutional Neural Networks

Pooling

- Pooling reduces spatial dimensions, providing:
  - Translation invariance
  - Reduced computation
  - Larger receptive field
- Example - **Max** Pool = (3x3) with stride = 3

$$\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 10 & 0 \\ 10 & 0 \end{bmatrix}$$

Convolutional Neural Networks

Pooling

- Pooling reduces spatial dimensions, providing:
  - Translation invariance
  - Reduced computation
  - Larger receptive field
- Example - **Average** Pool = (3x3) with stride = 3

$$\begin{bmatrix} 1 & 10 & 1 & 0 & 1 & 0 \\ 1 & 10 & 1 & 0 & 1 & 0 \\ 1 & 10 & 1 & 0 & 1 & 0 \\ 10 & 0 & 0 & 1 & 0 & 0 \\ 10 & 0 & 0 & 1 & 0 & 0 \\ 10 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 4 & 0.3 \\ 3.3 & 0.3 \end{bmatrix}$$

You can think of average pooling as applying a fixed blur filter

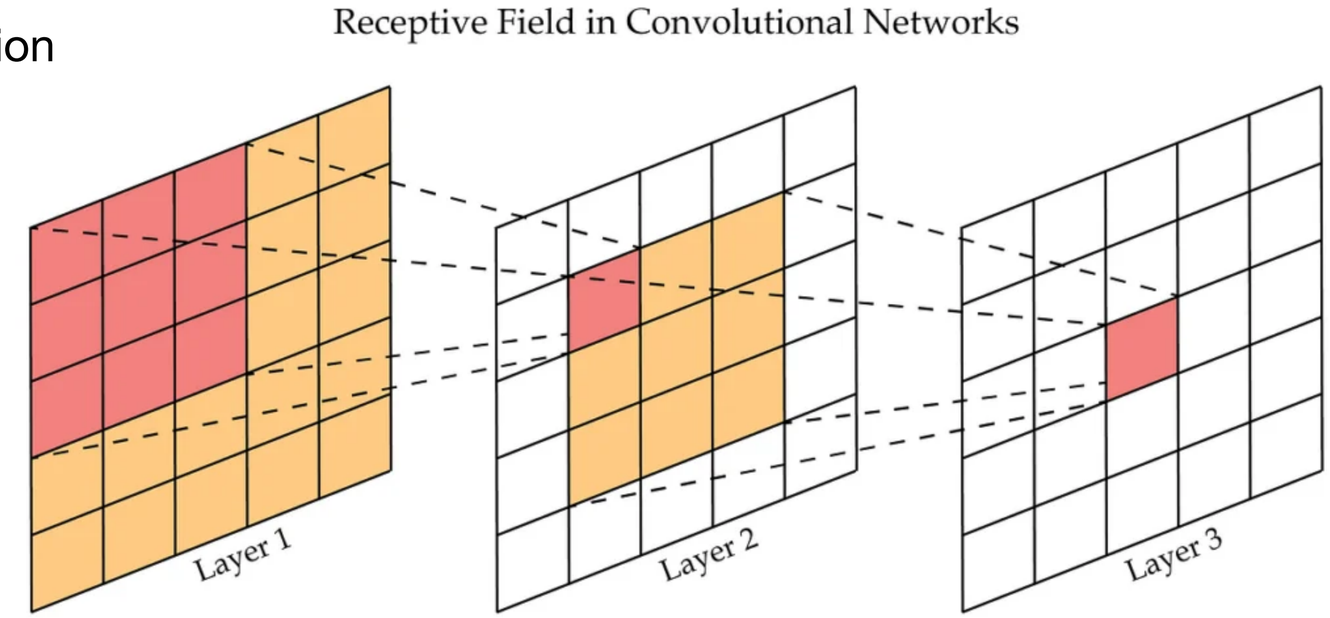
$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$ — Blur (Averaging)

Convolutional Neural Networks

Pooling

- The receptive field is the **region of the input** that influences a particular output neuron.
- With stacking, receptive fields grow:
  - Layer 1 (3×3 filter): Each filter sees 3×3 input region
  - Layer 2 (3×3 filter): Each filter sees 5×5 input region
  - Layer 3 (3×3 filter): Each filter sees 7×7 input region

$$\begin{bmatrix} 1 & 10 & 1 & 0 & 1 & 0 \\ 1 & 10 & 1 & 0 & 1 & 0 \\ 1 & 10 & 1 & 0 & 1 & 0 \\ 10 & 0 & 0 & 1 & 0 & 0 \\ 10 & 0 & 0 & 1 & 0 & 0 \\ 10 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 4 & 0.3 \\ 3.3 & 0.3 \end{bmatrix}$$



Convolutional Neural Networks

Pooling

Why it matters: Deeper layers have larger receptive fields, allowing them to detect larger, more complex patterns.

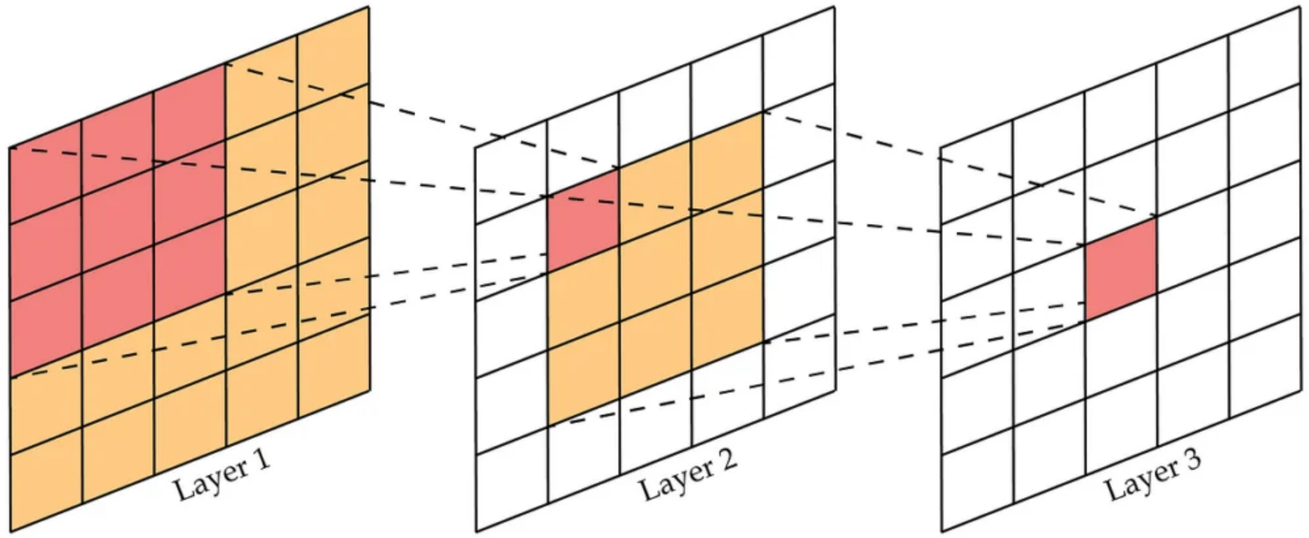
- The receptive field is the **region of the input** that influences a particular output neuron.
- With stacking, receptive fields grow:
  - Layer 1 (3×3 filter): Each filter sees 3×3 input region
  - Layer 2 (3×3 filter): Each filter sees 5×5 input region
  - Layer 3 (3×3 filter): Each filter sees 7×7 input region

Formula for L layers of $k \times k$ filters with stride 1:

$$RF = L \times (k - 1) + 1$$

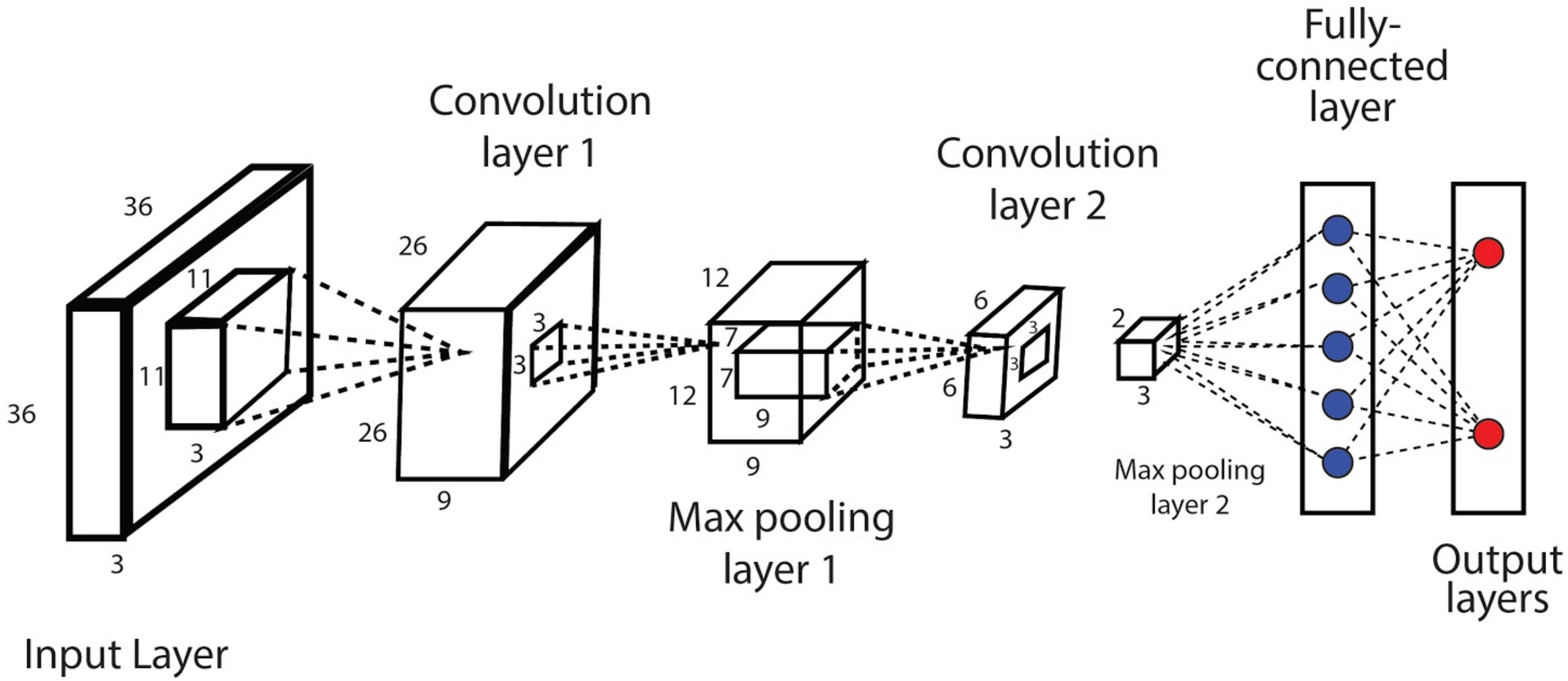
$$\begin{bmatrix} 1 & 10 & 1 & 0 & 1 & 0 \\ 1 & 10 & 1 & 0 & 1 & 0 \\ 1 & 10 & 1 & 0 & 1 & 0 \\ 10 & 0 & 0 & 1 & 0 & 0 \\ 10 & 0 & 0 & 1 & 0 & 0 \\ 10 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 4 & 0.3 \\ 3.3 & 0.3 \end{bmatrix}$$

Receptive Field in Convolutional Networks



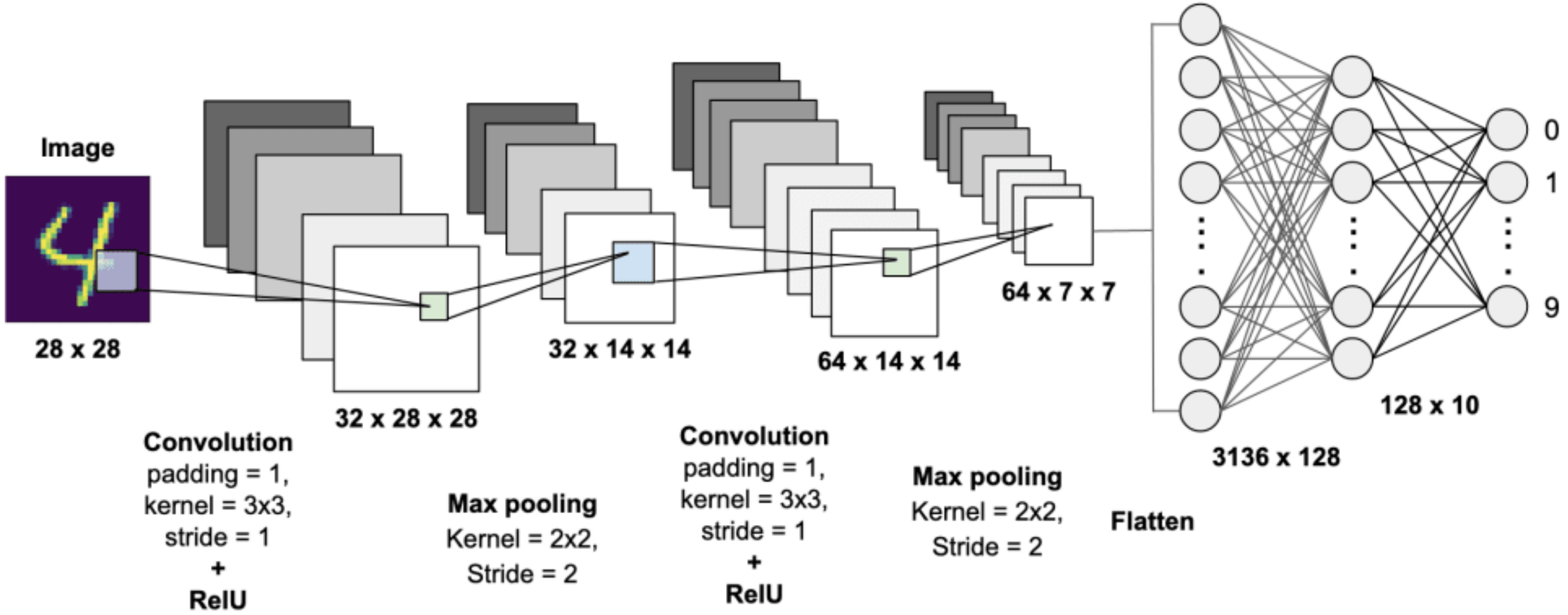
Convolutional Neural Networks

Overall Architecture



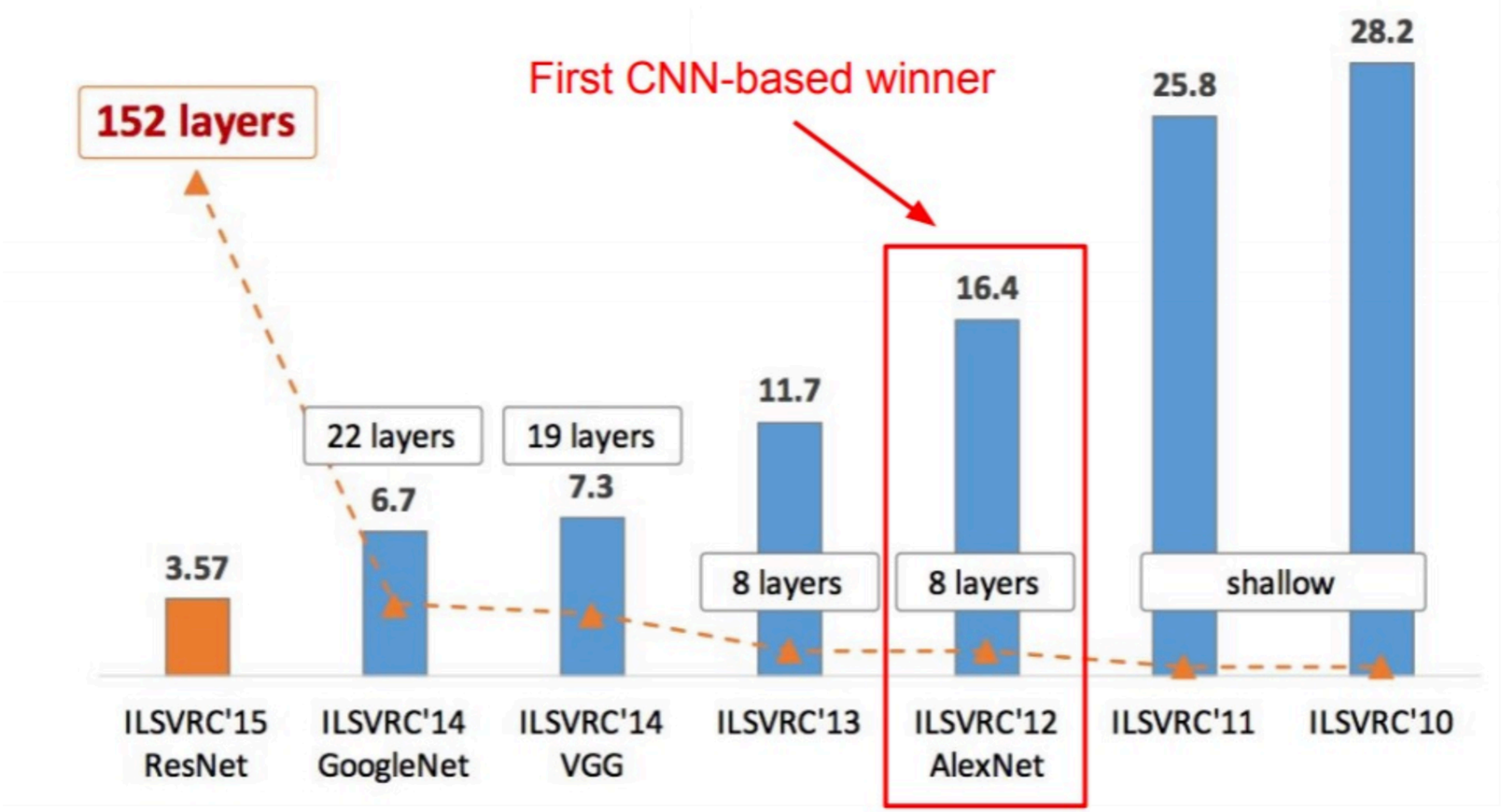
Convolutional Neural Networks

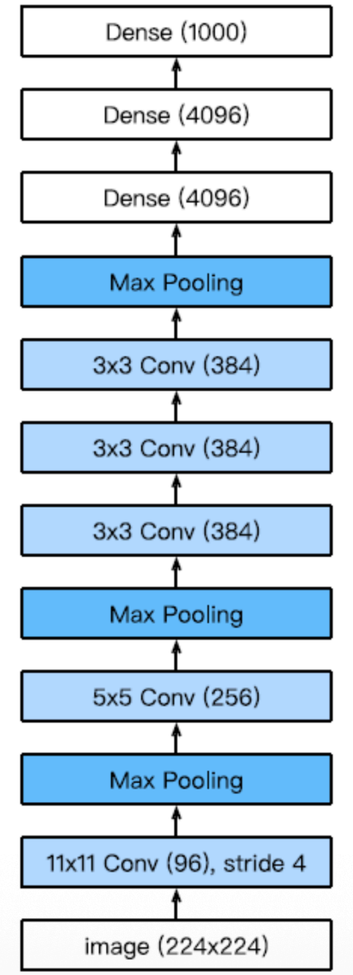
Overall Architecture



Historical Architectures

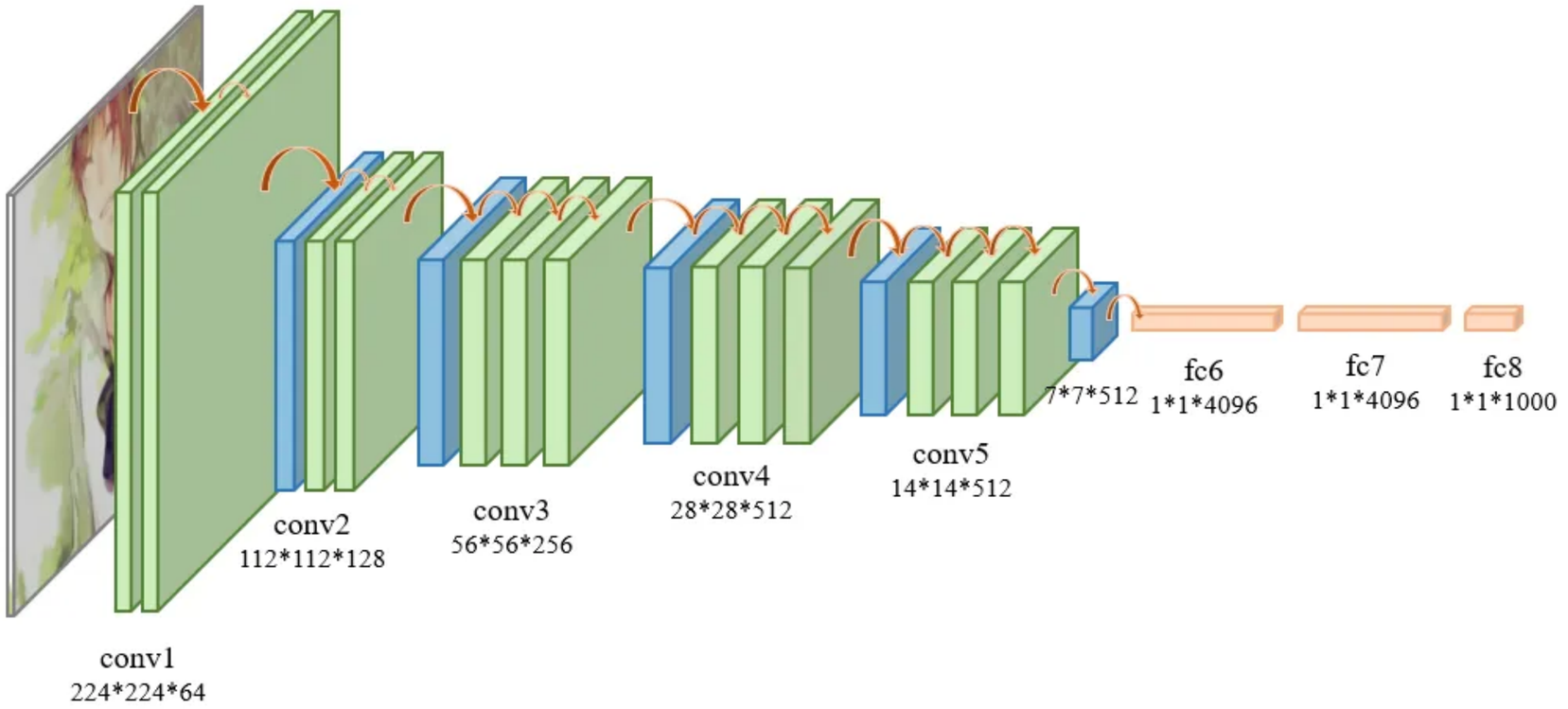
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) Winners





Historical Architectures

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) Winners



Data Augmentation for Images

Geometric Transforms

- **Random crop:** Crop random regions from larger image
- **Random horizontal flip**: P=0.5 usually
- **Random rotation:** Small angles (±15°)
- Random scale/zoom
- Random affine transforms

Data Augmentation for Images

Color Transforms

- **Color jitter:** Randomly adjust brightness, contrast, saturation, hue
- **Random grayscale:** Sometimes convert to grayscale
- Channel shuffle/drop

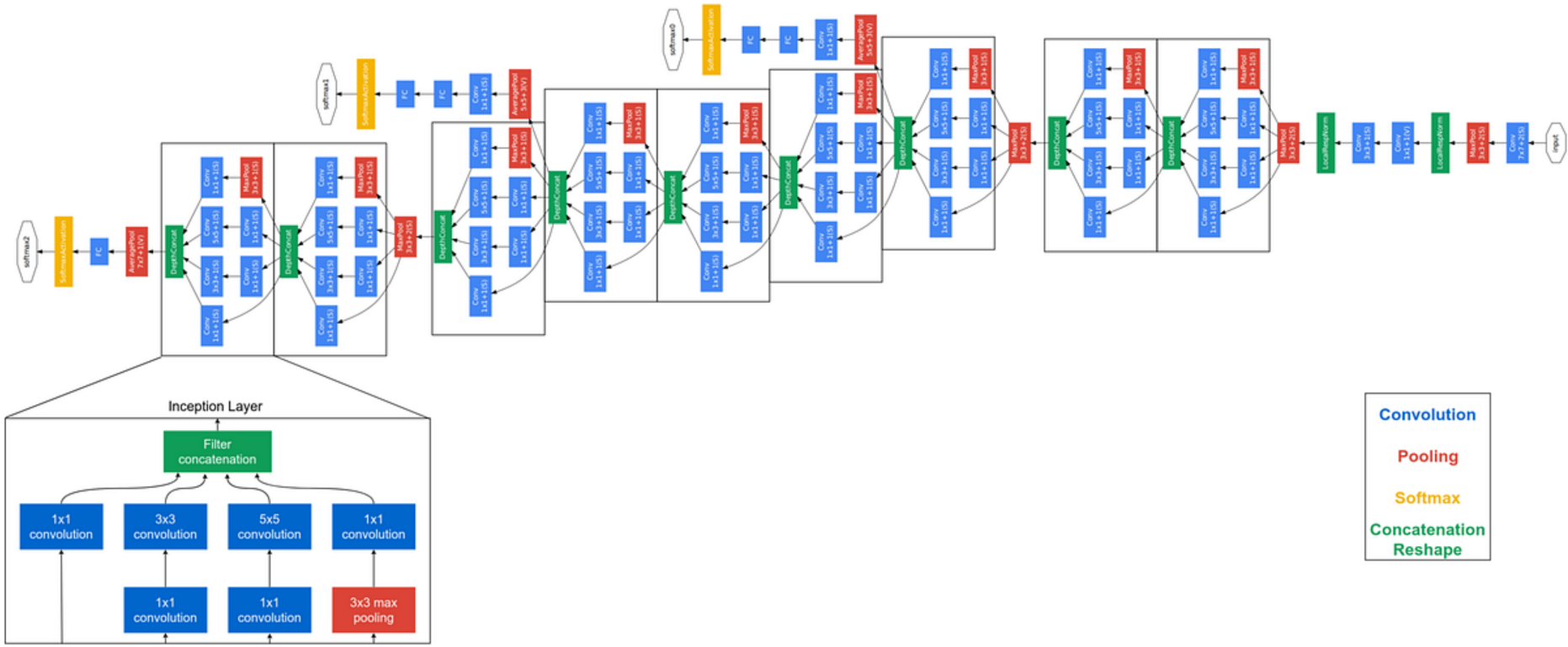
Data Augmentation for Images

Advanced Augmentations

- **Cutout:** Randomly mask square regions
- **Mixup:** Blend two images and their labels
- **CutMix:** Cut and paste patches between images

Transfer Learning

- Training from scratch requires:
  - Millions of labeled images
  - Days of GPU time
  - Expertise in architecture design

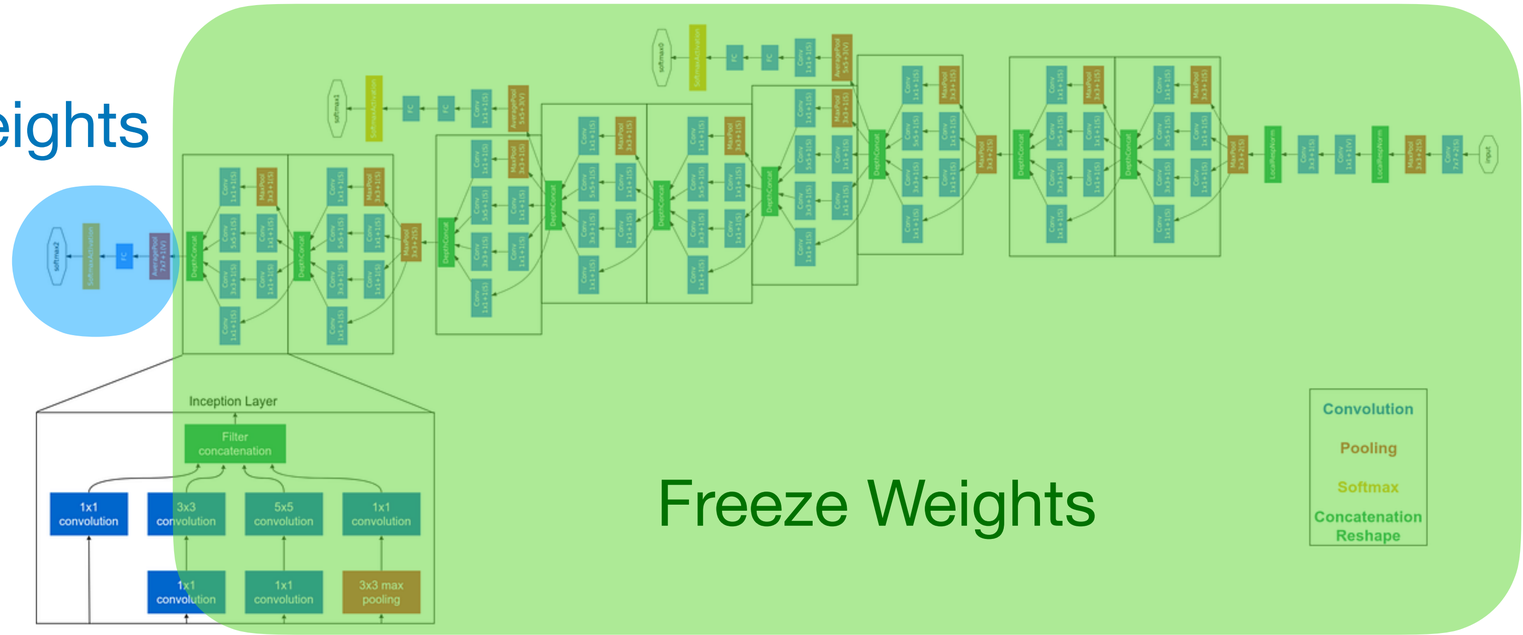


Transfer Learning

- GoogLeNet pre-trained models have learned:
  - Low-level features (edges, textures) - Layer 1-2
  - Mid-level features (shapes, parts) - Layer 3-4
  - High-level features (object parts) - Layer 5+
- These features are **transferable** to many vision tasks

Transfer Learning

- These features are **transferable** to many vision tasks
- Key Idea:
  - Freeze pretrained layers, train only new classifier

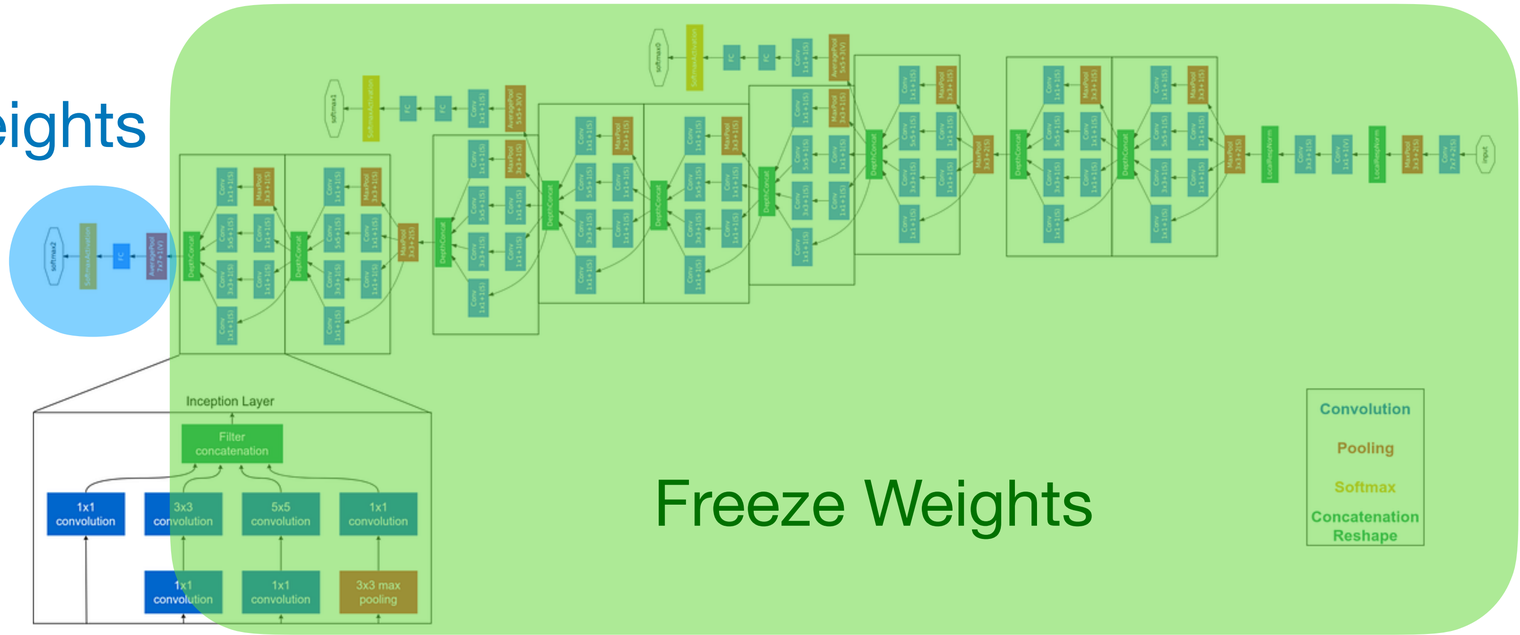


Transfer Learning

- These features are **transferable** to many vision tasks
- Key Idea:
  - Freeze pretrained layers, train only new classifier

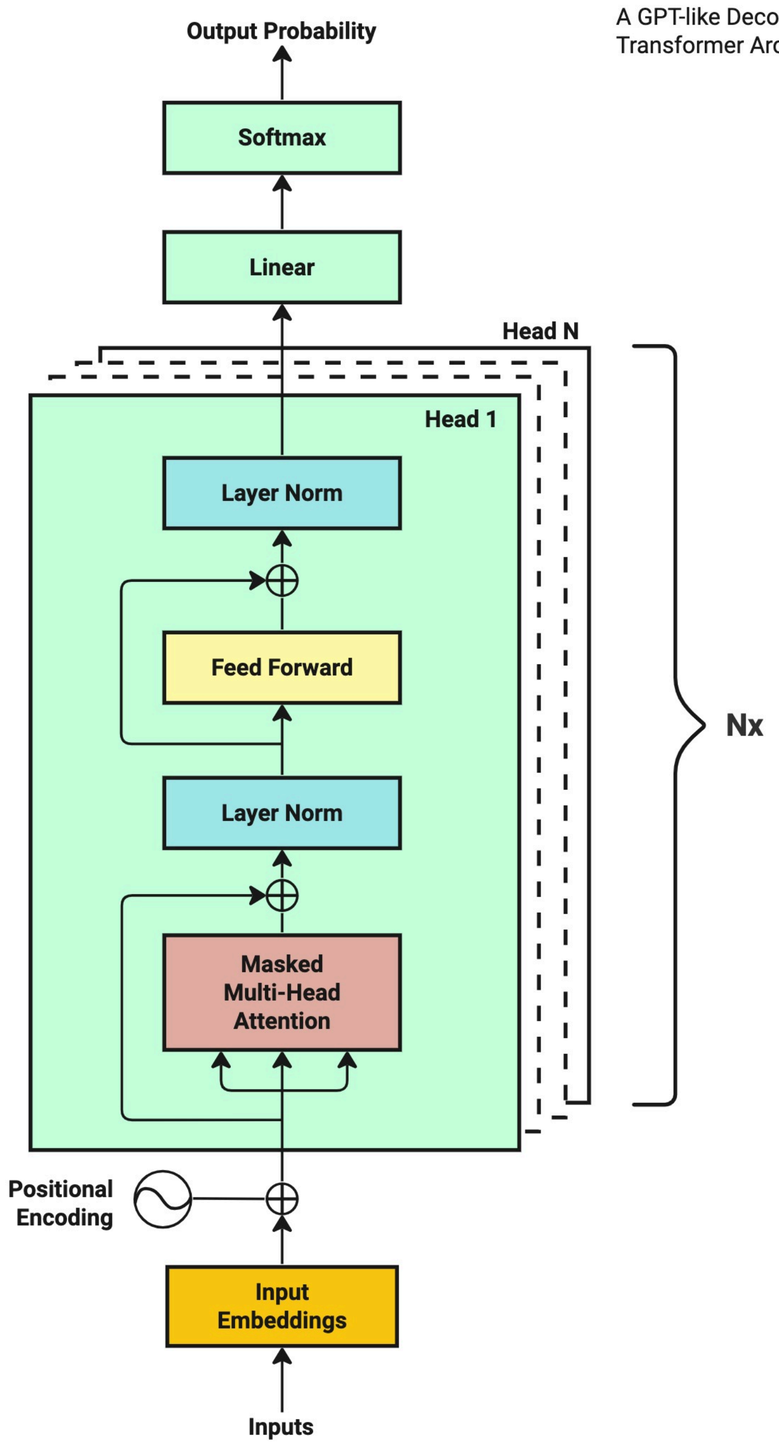
Fine-tuning Process

Re-train Weights



Transfer Learning

Current LLM Architecture

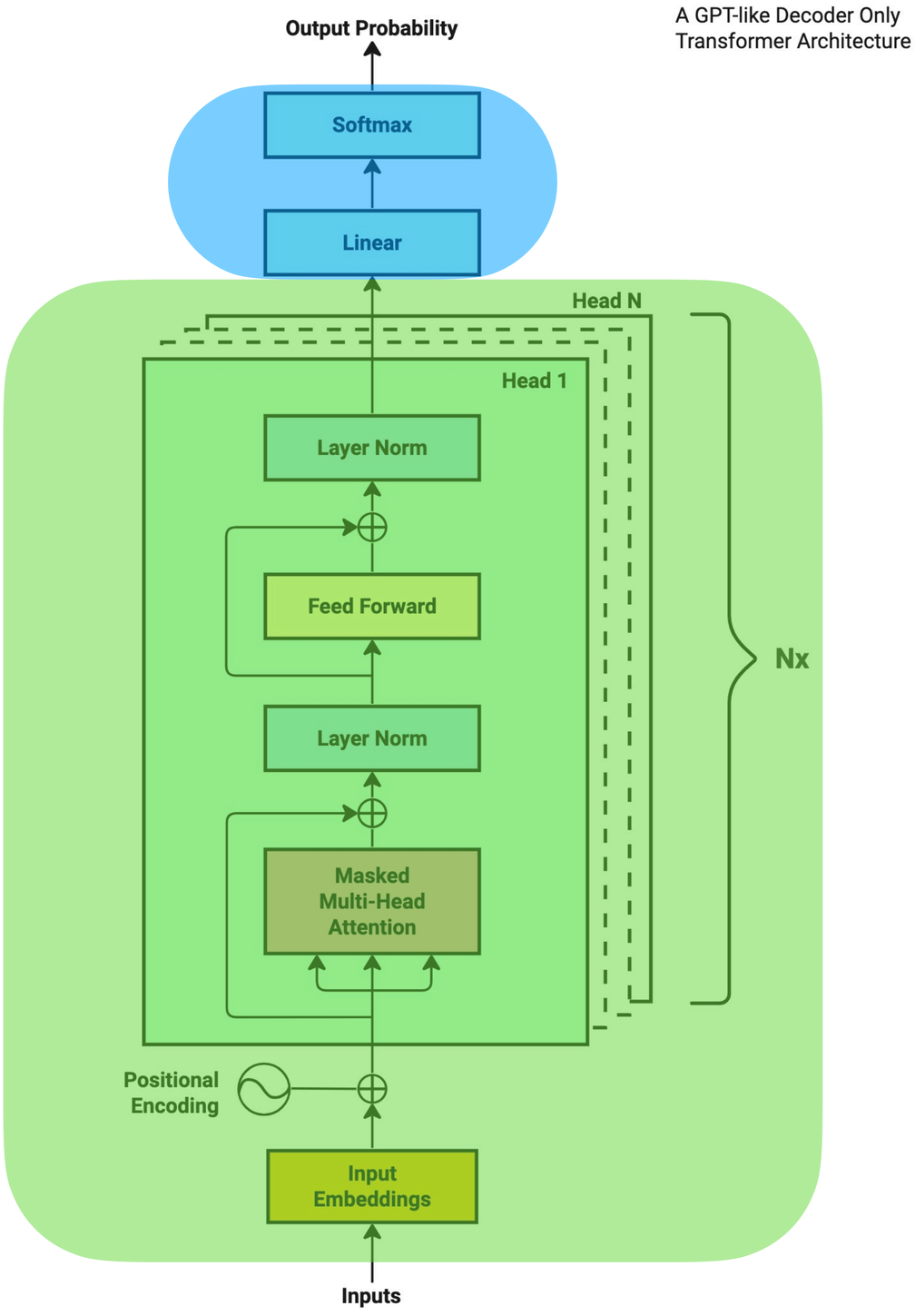


Transfer Learning

Current LLM Architecture

A very simplified fine-tuning approach:

1. Keep initial layers fixed
2. Retrain final prediction layers



Convolutional Neural Networks

- CNNs exploit these insights through:
  - **Local receptive fields**: Each neuron sees only a small region
  - **Weight sharing:** Same detector (filter) applied across entire image
  - **Pooling:** Reduces spatial dimensions, adds invariance

CNN Summary

- Applicable to data with natural grid topology
  - Time series
  - Images
- Convolution is a linear operation that uses local information
- Used for dimensionality reduction and learning hierarchical feature representations
- Much fewer parameters relative to Feed-Forward Neural Networks
- Reached human-level performance in ImageNet in 2014

Today’s Outline

- Convolutional Neural Networks
- Unsupervised Learning

Unsupervised Learning

- **Supervised Learning**: Learn from labeled data (X, y) to predict y for new X.
- **Unsupervised Learning**: Learn patterns from unlabeled data (X only)
  - No target variable y available
  - The goal is to discover hidden structure in data without explicit guidance.

Unsupervised Learning

- **Dimensionality Reduction**
  - PCA, t-SNE, UMAP, Autoencoders
- **Clustering**
  - K-Means, Hierarchical Clustering, Spectral Clustering
- **Representation Learning**
  - Autoencoders, VAE

Unsupervised Learning

- **Labeling is expensive**: Most real-world data is unlabeled
- **Exploratory analysis**: Discover structure before building models
- **Feature learning**: Learn representations for downstream tasks
- **Data compression**: Reduce storage and computation
- **Visualization**: Project high-dimensional data to 2D/3D

Principal Component Analysis (PCA)

Motivation

- High-dimensional data is hard to visualize, analyze, and model.
- PCA finds a lower-dimensional representation that **captures the most important information**.
- **Key Insight:** Not all dimensions are equally important.
  - Some directions in the data have high variance (informative), others have low variance (noise).

Principal Component Analysis (PCA)

Motivation

- PCA finds new axes (principal components) such that:
  - PC1: Direction of maximum variance
  - PC2: Direction of maximum remaining variance, **orthogonal** to PC1
  - PC3: Direction of maximum remaining variance, **orthogonal to PC1 and PC2**

Principal Component Analysis (PCA)

Formulation

- **Goal:** Find a linear transformation that projects data onto directions of maximum variance.
- Given data matrix $X \in \mathbb{R}^{n \times d}$ (n samples, d features)
  - Compute covariance matrix

$$C = \frac{1}{n-1} X^T X$$

  - Compute Eigen-decomposition of C

$$Cv_i = \lambda_i v_i$$

Principal Component Analysis (PCA)

Formulation

- Given data matrix $X \in \mathbb{R}^{n \times d}$ (n samples, d features)
  - Compute covariance matrix

$$C = \frac{1}{n-1} X^T X$$

  - Compute Eigen-decomposition of C

$$Cv_i = \lambda_i v_i$$

  - Sort Eigenvalues $\lambda_1 \geq \lambda_2 \ldots \geq \lambda_n$
  - Project into generated dimensions

$$Z = XW_k \text{ where } W_k = [v_1, v_2, \ldots v_k]$$

Principal Component Analysis (PCA)

Formulation

- Project into generated dimensions

$$Z = XW_k \text{ where } W_k = [v_1, v_2, \ldots v_k]$$

- How do you pick k, the number of dimensions to project to?
  - Each eigenvalue λ_i represents the variance captured by the i^{th} principal component.

$$\text{Proportion of variance explained} = \frac{\lambda_i}{\sum_{j=1}^{n} \lambda_j}$$

$$\text{Cumulative variance explained} = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{j=1}^{n} \lambda_j}$$

Rule of thumb: Keep enough components (k) to explain 90-95% of variance.

Principal Component Analysis (PCA)

Formulation

- Project into generated dimensions

$$Z = XW_k \text{ where } W_k = [v_1, v_2, \ldots v_k]$$

- How do you pick k, the number of dimensions to project to?
  - Each eigenvalue λ_i represents the variance captured by the i^{th} principal component.

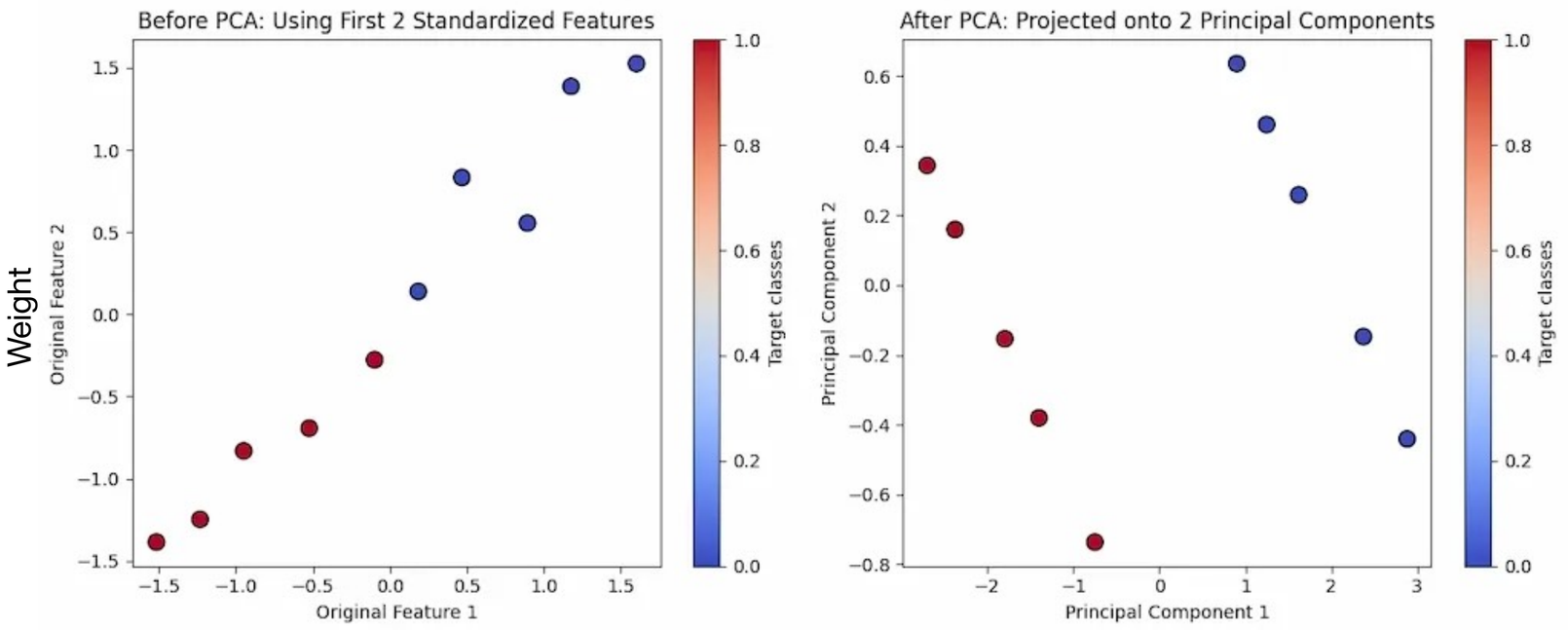
$$\text{Proportion of variance explained} = \frac{\lambda_i}{\sum_{j=1}^{n} \lambda_j}$$

$$\text{Cumulative variance explained} = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{j=1}^{n} \lambda_j}$$

Rule of thumb: Keep enough components (k) to explain 90-95% of variance.

Principal Component Analysis (PCA)

	Height	Weight	Age	Gender
0	170	65	30	1
1	165	59	25	0
2	180	75	35	1
3	175	68	28	1
4	160	55	22	0
5	172	70	32	1
6	168	62	27	0
7	177	74	33	1
8	162	58	24	0
9	158	54	21	0



Weight

Height

Principal Component Analysis (PCA)

Features, Pros and Cons

- Always make sure to normalize input features (say mean-variance normalization)
  - Without normalization: Features with larger variance dominate
  - With normalization: All features contribute equally

- Pros:
  - Multicollinearity Handling: Creates new, uncorrelated variables to address issues when **original features are highly correlated**.
  - Noise Reduction: Eliminates components with low variance
  - Data Compression: Represents data with fewer components reduce storage needs and speeding up processing.
  - Outlier Detection: Identifies unusual data points by showing which ones deviate significantly in the reduced space.

- Cons:
  - Assumes linear relationships, can't capture non-linear structure
  - Maximum variance direction may not be most informative - variance $\neq$ importance
  - Outliers can dominate variance
  - Lack of interpretability - PCs are mixtures of original features

Non-linear Dimensionality Reduction

- Real data is often not linearly separable - need non-linear methods for dimensionality reduction
- Intuition behind t-SNE and UMAP

Non-linear Dimensionality Reduction

t-SNE (t-Distributed Stochastic Neighbor Embedding)

- **Goal:** Preserve local neighborhood structure in low-dimensional embedding.
  - **Step 1:** Compute pairwise similarities in high-dimensional space (P)
  - **Step 2:** Define similarities in low-dimensional space using t-distribution (Q)
  - **Step 3:** Minimize KL divergence between P and Q

Non-linear Dimensionality Reduction

t-SNE (t-Distributed Stochastic Neighbor Embedding)

- **Goal:** Preserve local neighborhood structure in low-dimensional embedding.
  - **Step 1:** Compute pairwise similarities in high-dimensional space (P)

$$p_{ij} = \frac{exp(-\frac{\|x_i - x_j\|}{2\sigma_i^2})}{\sum_{k \neq i} exp(-\frac{\|x_i - x_k\|}{2\sigma_i^2})}$$

then, symmetrize as $p_{ij} = \frac{p_{ij} + p_{ji}}{2n}$

Non-linear Dimensionality Reduction

t-SNE (t-Distributed Stochastic Neighbor Embedding)

- **Goal:** Preserve local neighborhood structure in low-dimensional embedding.
  - **Step 2:** Define similarities in low-dimensional space using t-distribution (Q)

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq l} (1 + \|y_k - y_l\|^2)^{-1}}$$

Non-linear Dimensionality Reduction

t-SNE (t-Distributed Stochastic Neighbor Embedding)

- **Goal:** Preserve local neighborhood structure in low-dimensional embedding.
  - **Step 3:** Minimize KL divergence between P and Q

$$KL(P\|Q) = \sum_{i \neq j} p_{ij} log \frac{p_{ij}}{q_{ij}}$$

Use gradient descent to minimize KL Divergence and find optimal y

Non-linear Dimensionality Reduction

t-SNE (t-Distributed Stochastic Neighbor Embedding)

- **Limitations:**
  - **Stochastic:** Different runs give different results
  - **Slow:** $O(n^2)$ complexity
  - **No new data:** Can't project new points without rerunning
  - **Distances not meaningful:** Only topology is preserved
  - Sensitive to hyperparameters

Non-linear Dimensionality Reduction

UMAP (Uniform Manifold Approximation and Projection)

- **Goal:** Similar to t-SNE, but faster and better preserves global structure.
- **Step 1:** Construct a weighted k-nearest neighbor graph in high-dimension
  - Connect each point to its **k nearest neighbors**
  - Weight edges by distance

Picking k:
Higher = more global structure
Lower = more local detail

Non-linear Dimensionality Reduction

UMAP (Uniform Manifold Approximation and Projection)

- **Goal:** Similar to t-SNE, but faster and better preserves global structure.
- **Step 1:** Construct a weighted k-nearest neighbor graph in high-dimension
  - Connect each point to its **k nearest neighbors**
  - Weight edges by distance

Picking k:
Higher = more global structure
Lower = more local detail

- **Step 2:** Construct a similar graph in low-dimensions

Non-linear Dimensionality Reduction

UMAP (Uniform Manifold Approximation and Projection)

- **Goal:** Similar to t-SNE, but faster and better preserves global structure.
- **Step 1:** Construct a weighted k-nearest neighbor graph in high-dimension
  - Connect each point to its **k nearest neighbors**
  - Weight edges by distance

Picking k:
Higher = more global structure
Lower = more local detail

- **Step 2:** Construct a similar graph in low-dimensions
- **Step 3:** Optimize low-dimensional positions to match the high-dimensional graph structure

Non-linear Dimensionality Reduction

UMAP (Uniform Manifold Approximation and Projection)

- **Goal:** Similar to t-SNE, but faster and better preserves global structure.
- **Step 1:** Construct a weighted k-nearest neighbor graph in high-dimension
  - Connect each point to its **k nearest neighbors**
  - Weight edges by distance

Picking k:
Higher = more global structure
Lower = more local detail

- **Step 2:** Construct a similar graph in low-dimensions
- **Step 3:** Optimize low-dimensional positions to match the high-dimensional graph structure
- Uses **cross-entropy loss** instead of KL divergence
- Includes both **attractive** and **repulsive** forces

Non-linear Dimensionality Reduction

UMAP (Uniform Manifold Approximation and Projection)

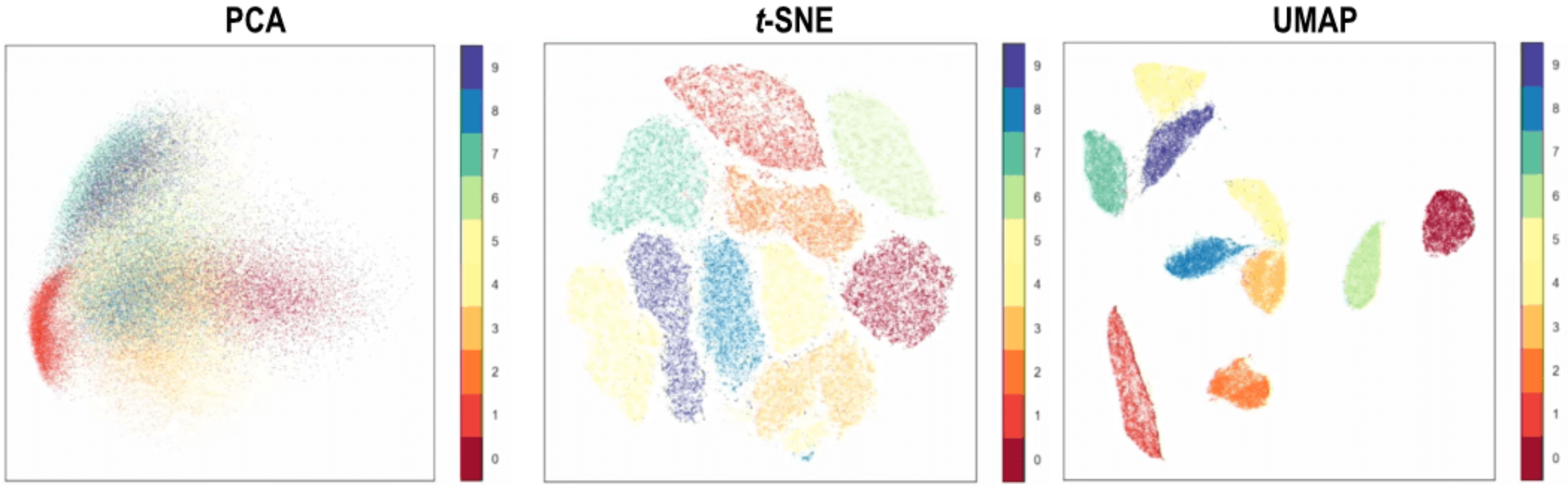
- **Goal:** Similar to t-SNE, but faster and better preserves global structure.
- **Step 1:** Construct a weighted k-nearest neighbor graph in high-dimension
  - Connect each point to its **k nearest neighbors**
  - Weight edges by distance

Picking k:
Higher = more global structure
Lower = more local detail

- **Step 2:** Construct a similar graph in low-dimensions
- **Step 3:** Optimize low-dimensional positions to match the high-dimensional graph structure
- Uses **cross-entropy loss** instead of KL divergence
- Includes both **attractive** and **repulsive** forces

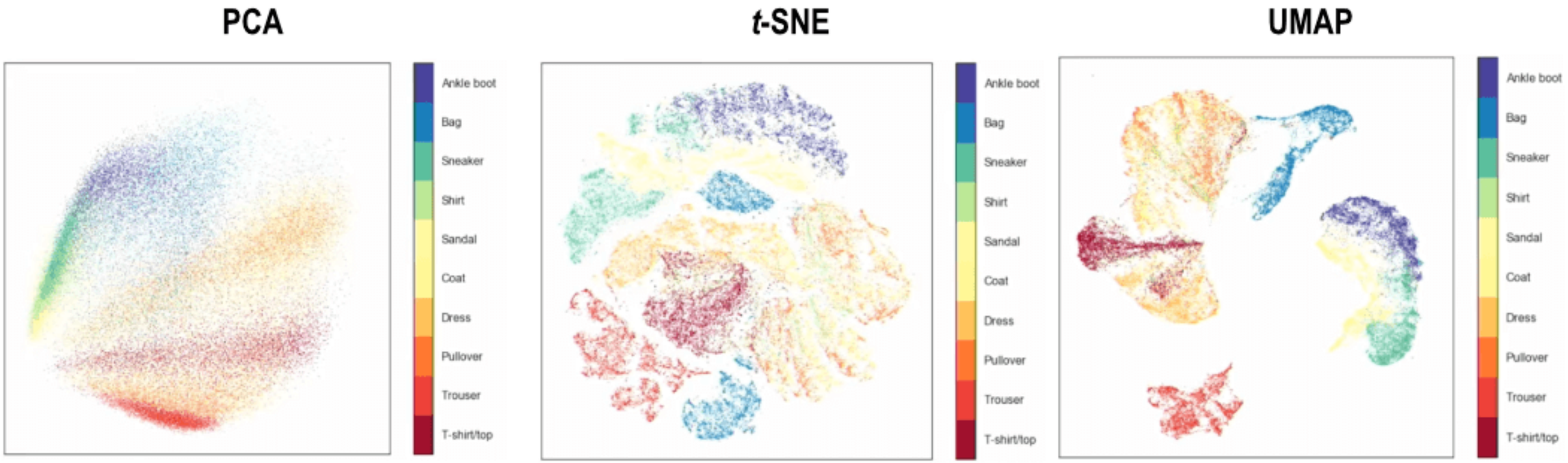
Dimensionality Reduction

MNIST Digits Dataset



Dimensionality Reduction

Fashion MNIST Dataset



Dimensionality Reduction

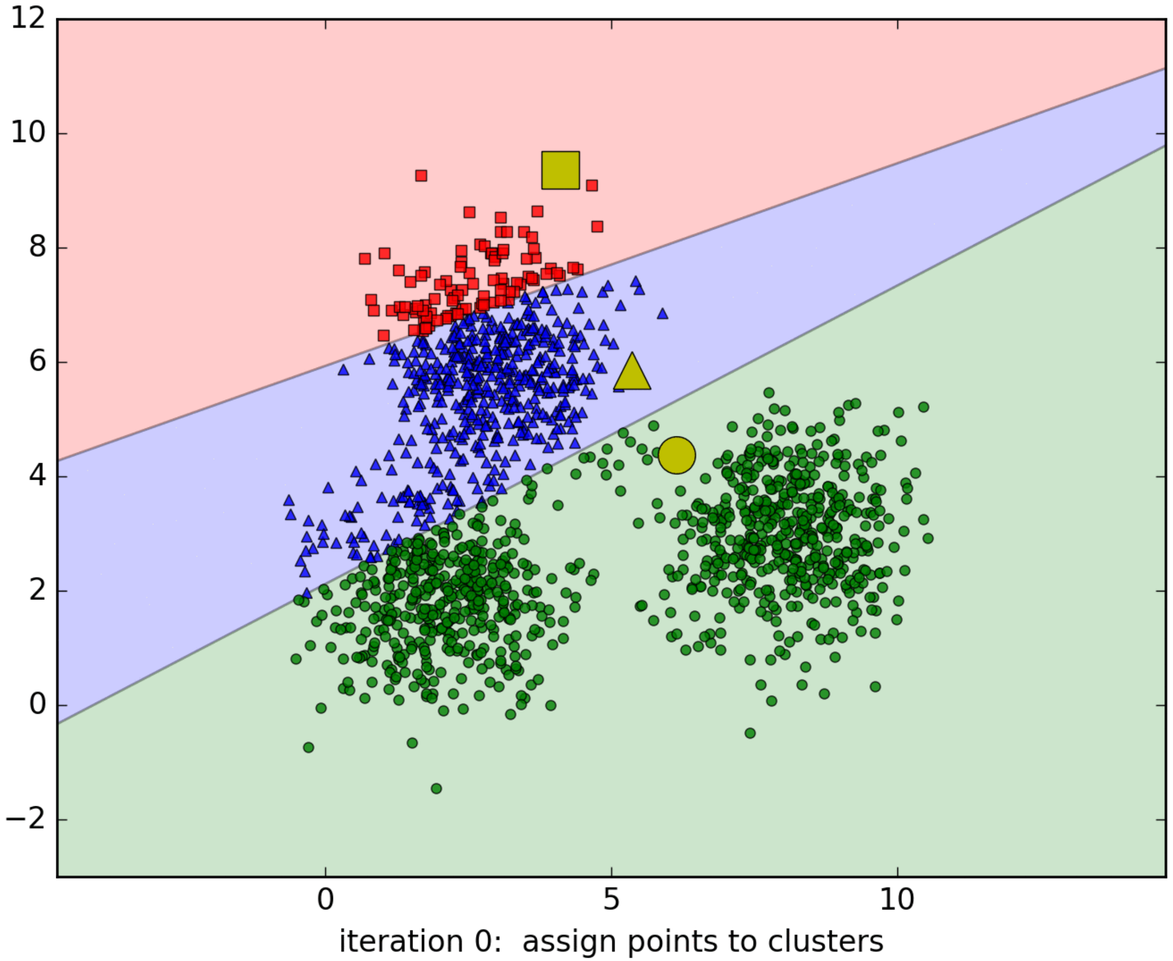
Metric	PCA	t-SNE	UMAP
Type	Linear	Non-Linear	Non-Linear
Preserves	Global Variance	Local Neighborhoods	Local + Global
Speed	Very Fast	Slow	Fast
Deterministic	Yes	No	Consistent but not deterministic
Interpretable	Somewhat	No	No
New Data	Yes	No	Yes
Use Cases	Preprocessing, quick visualization	Visualization	Visualization

Clustering

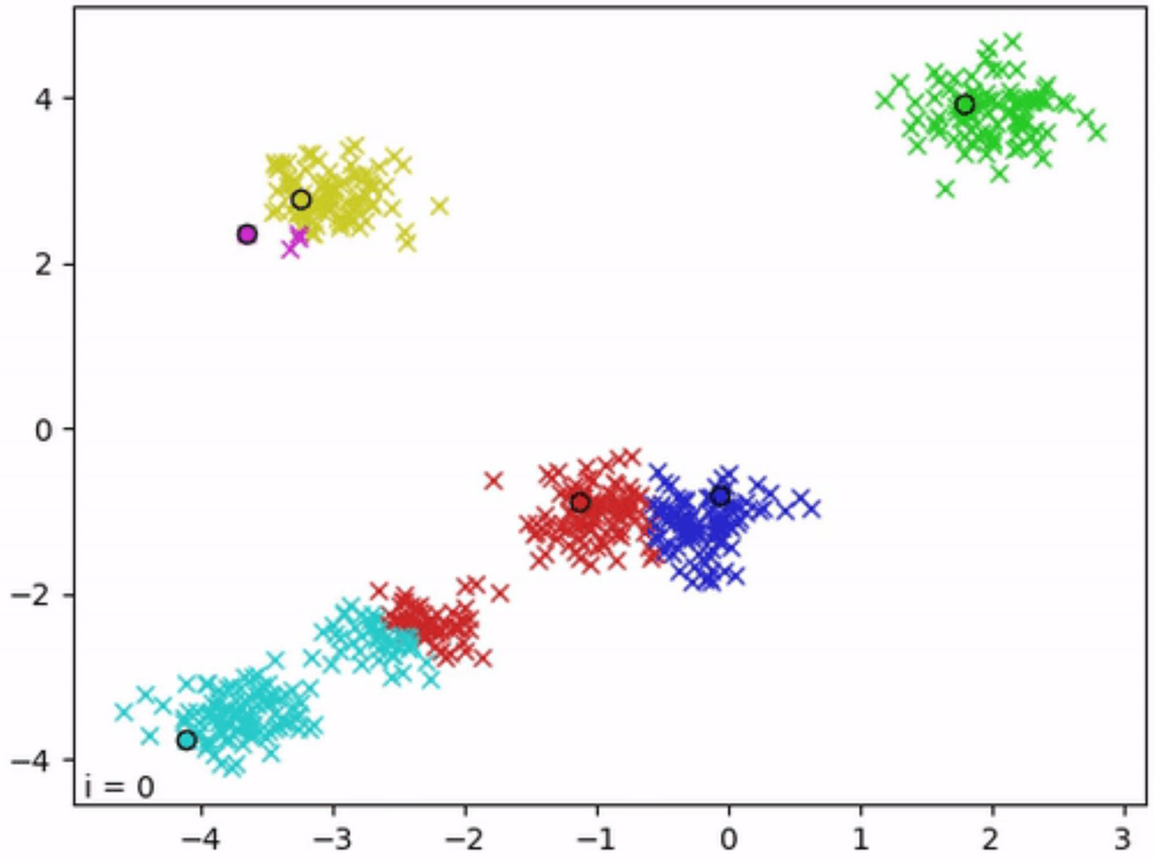
- **Goal:** Partition data into groups (clusters) such that:
  - Points within a cluster are similar
  - Points in different clusters are dissimilar
  - Unlike classification, **clusters are discovered**, not predefined.

Clustering

K-Means Clustering



iteration 0: assign points to clusters



Clustering

K-Mean Clustering

- **Input:** Data X, number of clusters k
- Randomly initialize k cluster centroids $\mu_1, \mu_2, \ldots, \mu_k$
- Repeat until convergence:
  - **ASSIGN:** For each point x_i, assign to nearest centroid μ

$$c_i = arg\min_j \|x_i - \mu_j\|^2$$

  - **UPDATE**: Recompute centroids as cluster means

$$\mu_j = \frac{1}{|C_j|}\sum_{i \in C_j} x_i$$

- **Return** cluster assignments and centroids

Clustering
K-Mean Clustering

- **Input:** Data X, number of clusters k
- Randomly initialize k cluster centroids $\mu_1, \mu_2, \ldots, \mu_k$
- Repeat until convergence:
  - **ASSIGN:** For each point x_i, assign to nearest centroid μ

$$c_i = arg\min_j \|x_i - \mu_j\|^2$$

  - **UPDATE**: Recompute centroids as cluster means

$$\mu_j = \frac{1}{|C_j|}\sum_{i \in C_j} x_i$$

- **Return** cluster assignments and centroids

K-Means minimizes the **within-cluster sum of squares**

$$L = \sum_{j=1}^{k}\sum_{i \in C_j} \|x_i - \mu_j\|^2$$

Each iteration is guaranteed to decrease (or maintain) L, so the algorithm converges.

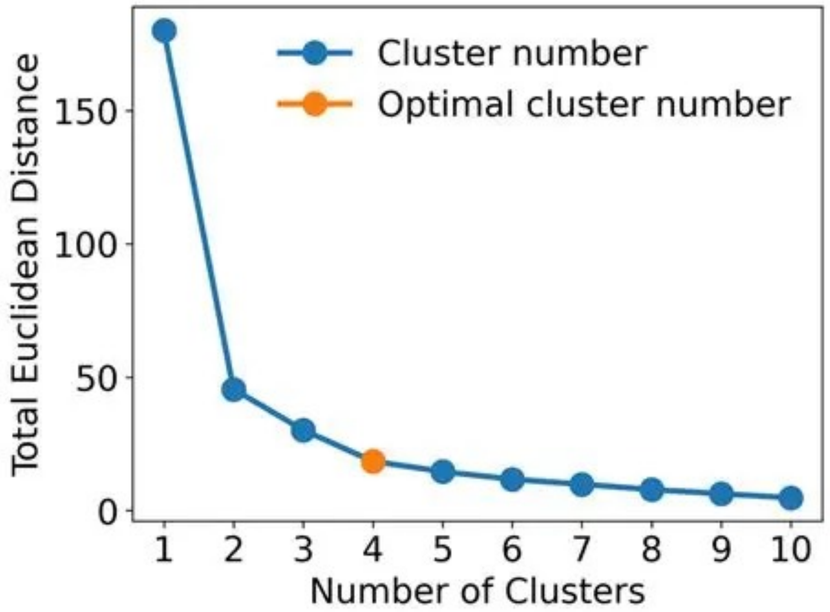
Guaranteed to converge (finite number of possible assignments)

NOT guaranteed to find global optimum (converges to local minimum)

Solution depends on **initialization**

Clustering

K-Mean Clustering



- Choosing k
  - Elow Method

Plot the loss $L = \sum_{j=1}^{k} \sum_{i \in C_j} \|x_i - \mu_j\|^2$ with increasing number of k and look for an “elbow” where adding more centroids gives diminishing returns

Clustering

K-Mean Clustering - Limitations

- Must specify k - need to know number of clusters in advance
- Assumes spherical/globular clusters - since it uses Euclidean distances, will likely fail for elongated or irregular shaped clusters
- Sensitive to outliers since they can pull centroids away
- Poor initialization of clusters can lead to poor local optima
- Sensitive to scale - need to ensure proper normalization

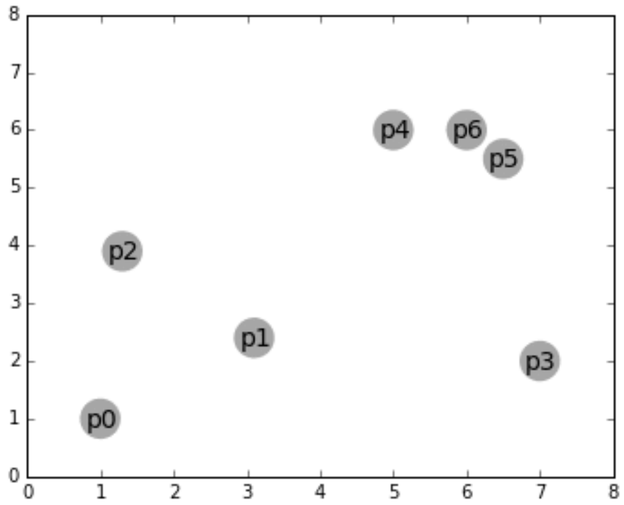
Clustering

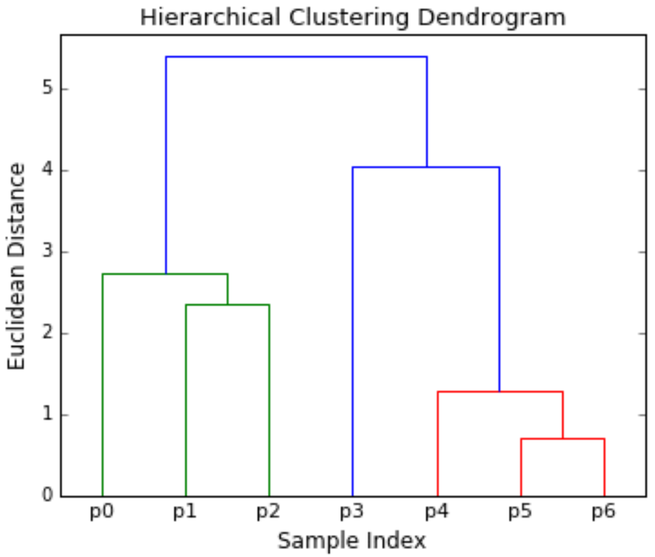
K-Mean Clustering - When to use?

- Good for:
  - Large datasets (scales well: $O(nkd)$ per iteration)
  - Roughly spherical, similar-sized clusters
  - When you know approximate number of clusters
- Not good for:
  - Non-convex cluster shapes
  - Clusters of very different sizes/densities
  - When number of clusters is unknown

Clustering

Hierarchical Clustering

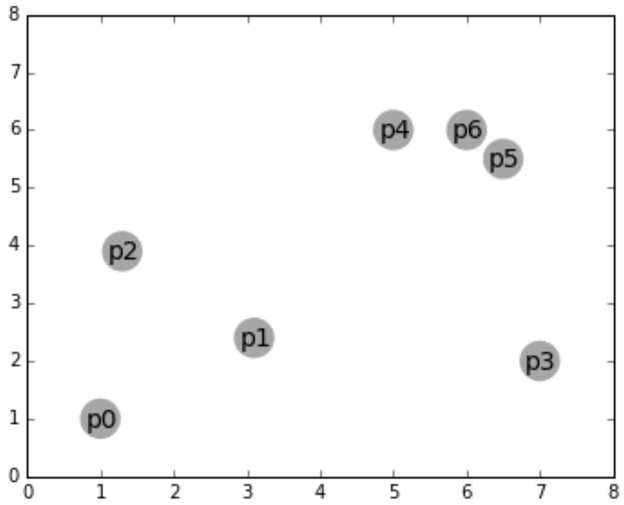


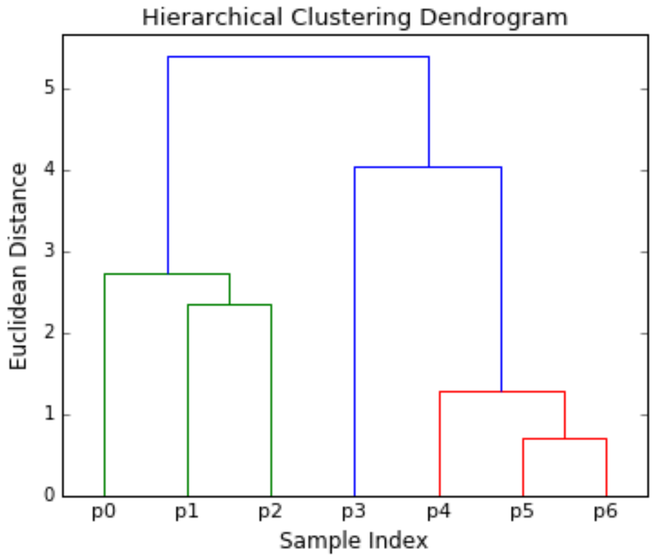


- Agglomerative (Bottom-up):
  - Start with each point as its own cluster
  - Merge closest pairs until one cluster remains.
- Divisive (Top-down):
  - Start with all points in one cluster
  - Recursively split until each point is its own cluster.
- Agglomerative is more common.

Clustering

Hierarchical Clustering



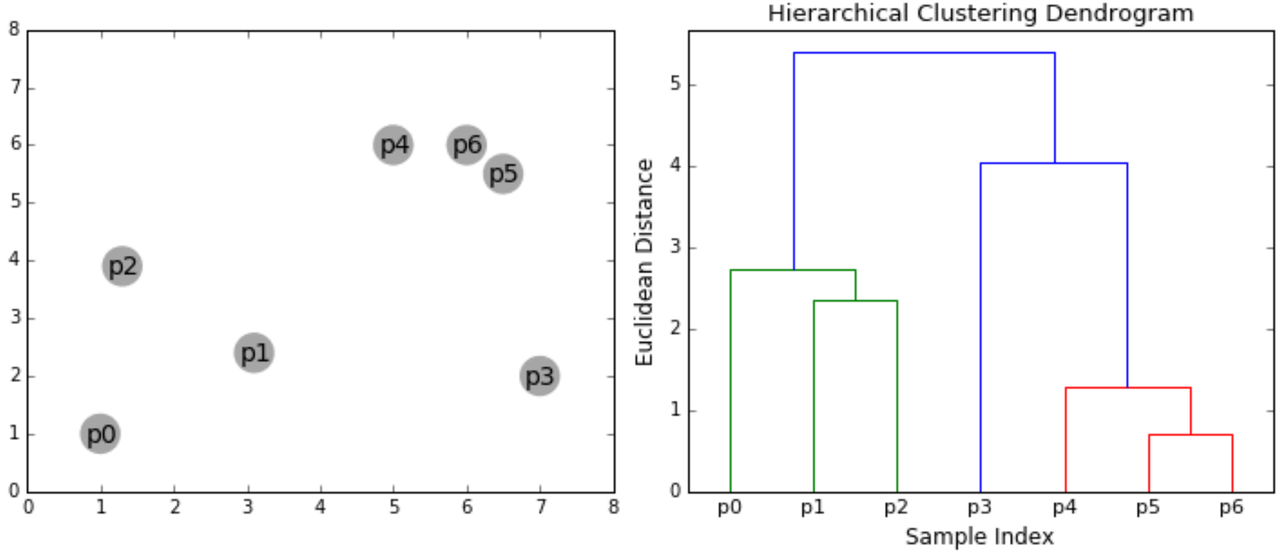


- Agglomerative Algorithm

1. Start: Each point is its own cluster (n clusters)
2. Compute distance between all pairs of clusters
3. Merge the two closest clusters
4. Repeat steps 2-3 until only one cluster remains
5. Record the merge history (dendrogram)

Clustering

Hierarchical Clustering



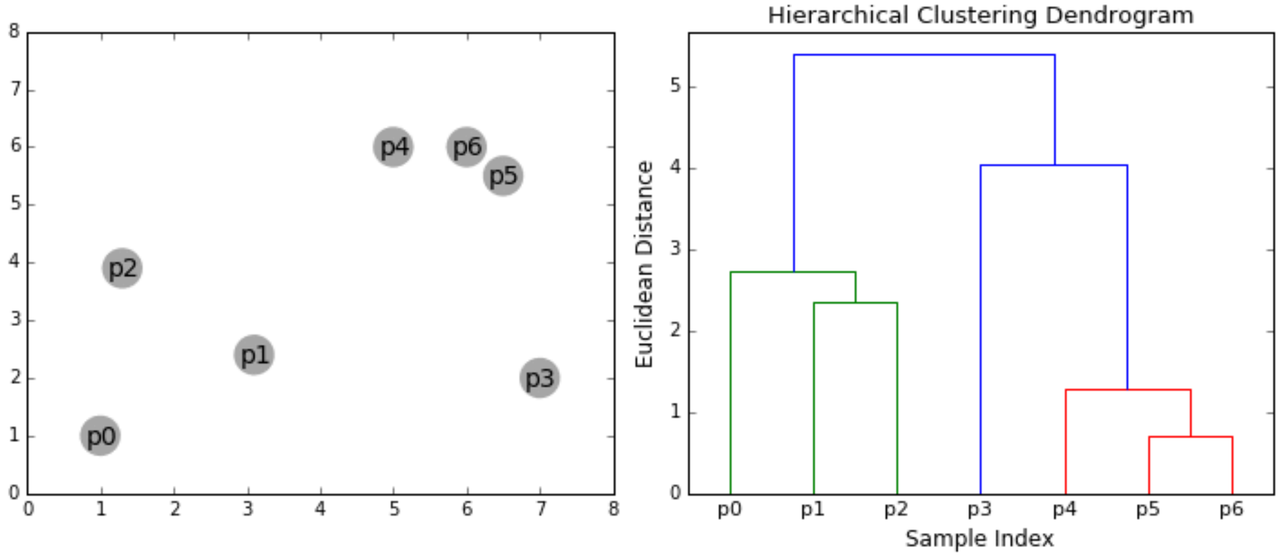
- Agglomerative Algorithm

  1. Start: Each point is its own cluster (n clusters)
  2. Compute distance between all pairs of clusters
  3. **Merge the two closest clusters**
  4. Repeat steps 2-3 until only one cluster remains
  5. Record the merge history (dendrogram)

Clustering

Hierarchical Clustering

- Linkage Criteria



Linkage	Definition	Properties
Single	Minimum distance between any two points	Can produce “chaining” - i.e., elongated clusters
Complete	Maximum distance between any two points in the clusters	Produces compact clusters
Average	Average distance between all pairs	Balance between single and complete

Clustering

Hierarchical Clustering - Pros and Cons

- Advantages:
  - No need to specify k in advance
  - Produces hierarchy (useful for taxonomy)
  - Dendrogram is informative
  - Any distance metric can be used
  - Deterministic (unlike K-Means)
- Disadvantages:
  - Slow: $O(n^3)$ time, $O(n^2)$space
  - Cannot “undo” a merge (greedy)
  - Sensitive to noise and outliers
  - Not suitable for large datasets

Clustering

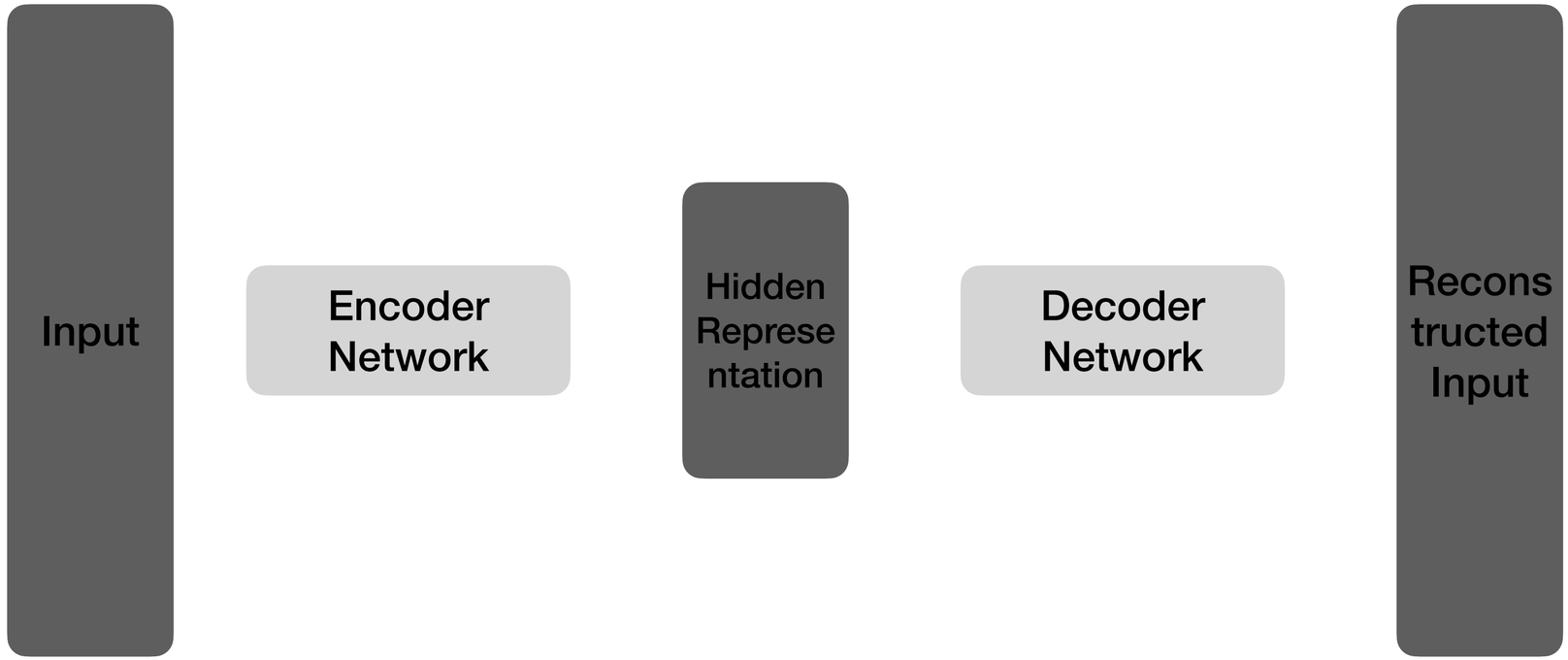
Summary

Algorithm	K-Means	Hierarchical Clustering
Cluster Shape	Spherical	Any
# Clusters	Must Specify	Automatic
Outliers	Sensitive	Sensitive
Scalability	Very Good	Poor
Use Cases	Large Data, Spherical Clusters	Small Data, Taxonomy, Interpretability

Autoencoders

Dimensionality Reduction using Deep Learning

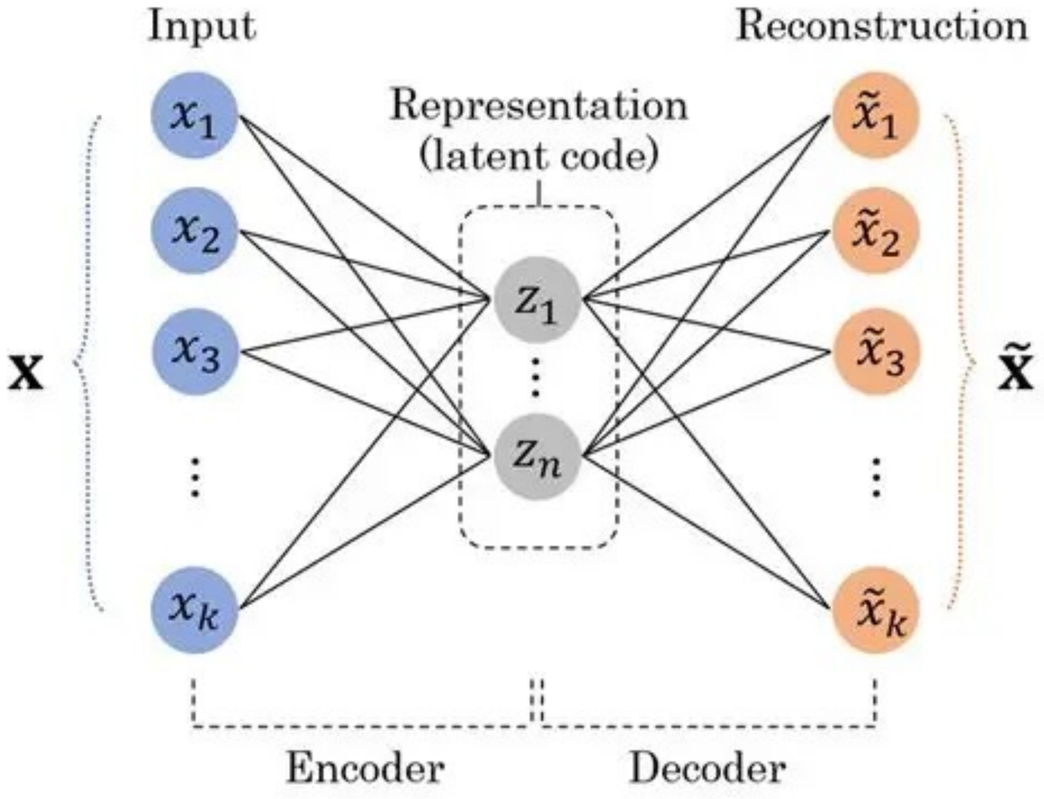
- An autoencoder is a neural network trained to **reconstruct** its input.

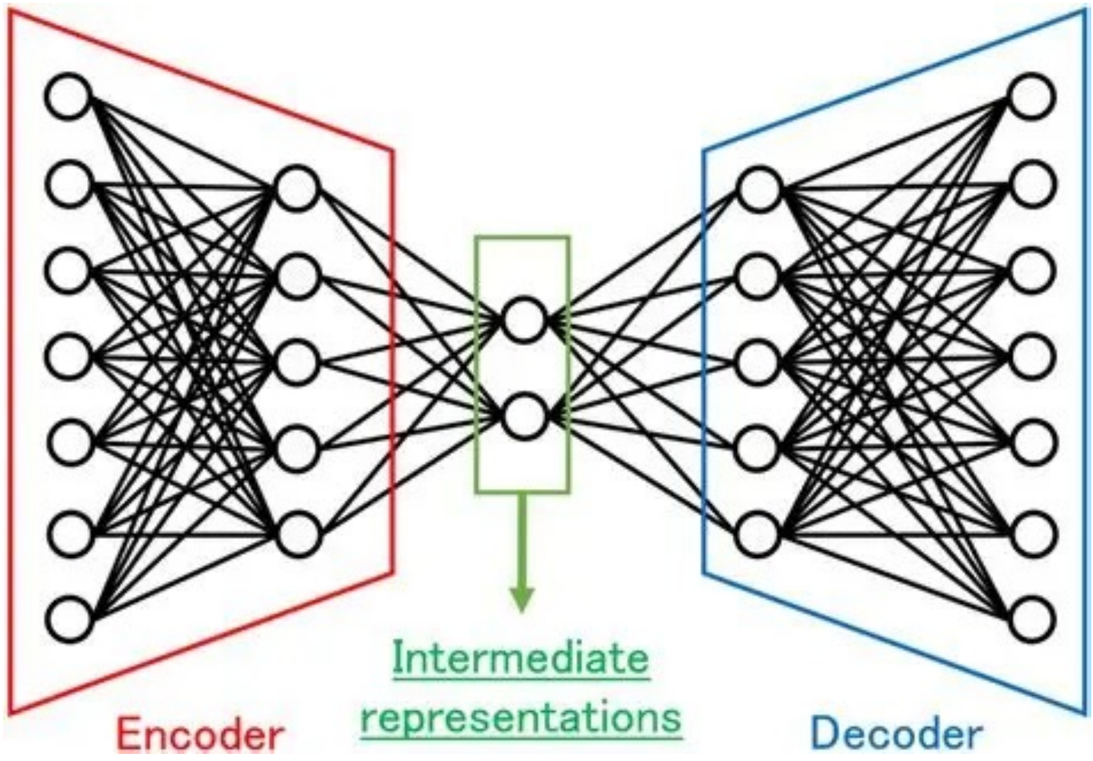


Autoencoders

Dimensionality Reduction using Deep Learning

- An autoencoder is a neural network trained to **reconstruct** its input.

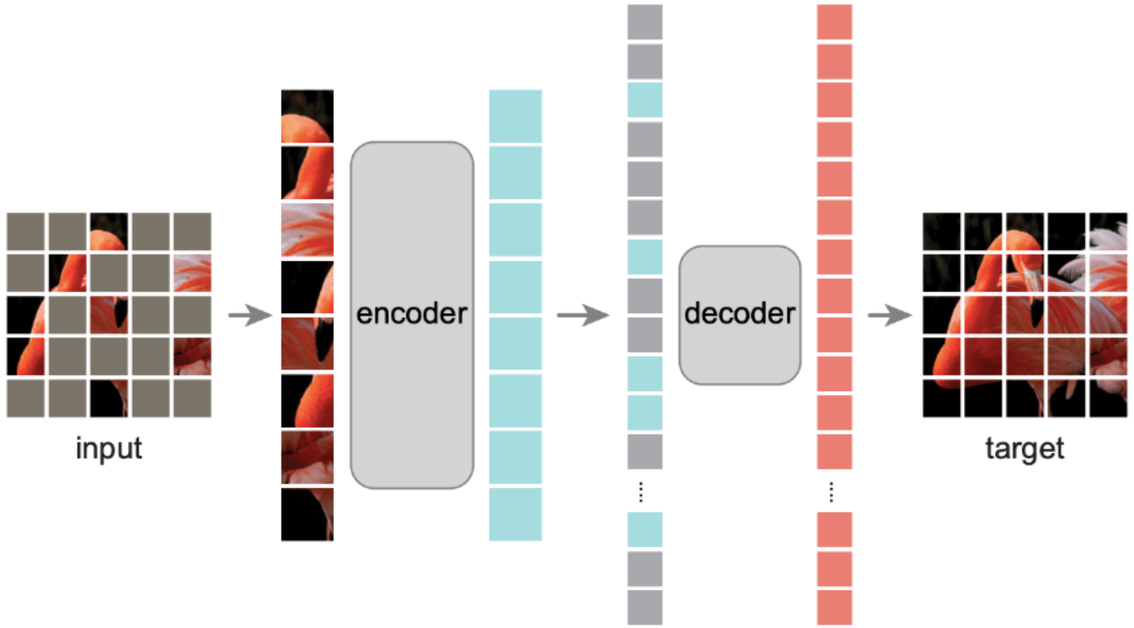


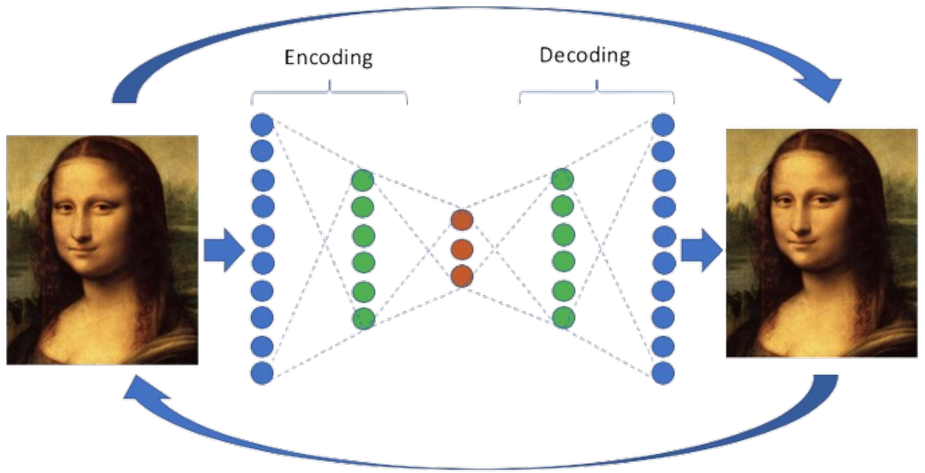


Autoencoders

Dimensionality Reduction using Deep Learning

- An autoencoder is a neural network trained to **reconstruct** its input.

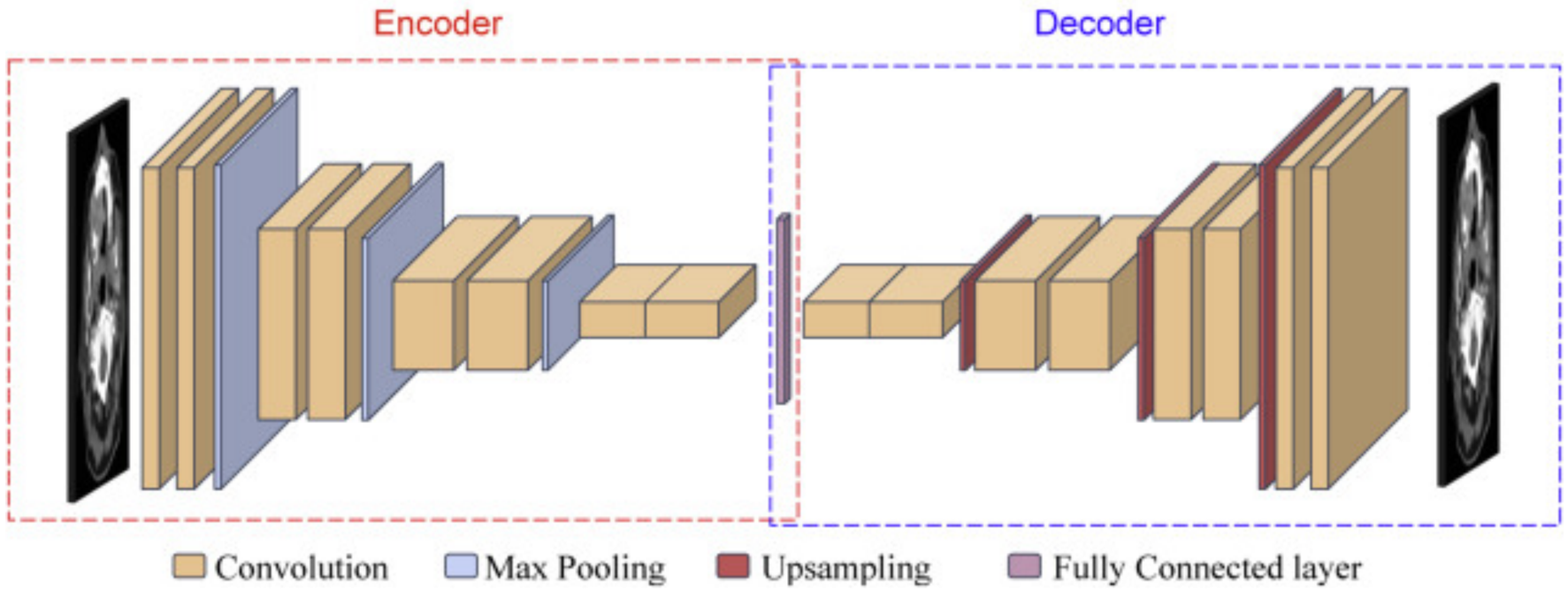




Autoencoders

Dimensionality Reduction using Deep Learning

- An autoencoder is a neural network trained to **reconstruct** its input.



Autoencoders

Dimensionality Reduction using Deep Learning

- An autoencoder is a neural network trained to **reconstruct** its input.
- **Encoder:** Compresses input to lower-dimensional latent representation
- **Latent space:** Compressed representation (bottleneck)
- **Decoder:** Reconstructs input from latent code
- **Training objective:** Minimize reconstruction error

$$L = ||x - \hat{x}||^2$$

Autoencoders

Dimensionality Reduction using Deep Learning

- Why does it work?
  - The **bottleneck** forces the network to learn a **compressed representation** that captures the most important features of the input.
  - The network **must**:
    - Learn which features are essential (encoding)
    - Learn how to reconstruct from those features (decoding)
  - This is similar to PCA, but autoencoders can learn non-linear transformations.

Autoencoders

Dimensionality Reduction using Deep Learning

- Denoising Autoencoder
  - **Corrupt** the input, train to reconstruct the **clean** version.
  - Corruption types: Gaussian noise, masking (dropout), salt-and-pepper noise

Autoencoders

Dimensionality Reduction using Deep Learning

- Applications of Autoencoders
  - **Dimensionality reduction**: Use encoder output as features
  - **Denoising**: Remove noise from images/signals
  - **Anomaly detection**: High reconstruction error = anomaly
  - **Pretraining**: Initialize deep networks
  - **Image compression**: Encode images compactly
  - **Feature learning**: Learn representations for downstream tasks

Variational Autoencoders

Dimensionality Reduction using Deep Learning

- Standard autoencoders learn a **deterministic** mapping to latent space (this is true for all neural networks in general)
- The latent space may have “holes”, i.e., regions that don't correspond to valid data.
- You can't sample from it to generate new data.

Variational Autoencoders

Dimensionality Reduction using Deep Learning

- Standard autoencoders learn a **deterministic** mapping to latent space (this is true for all neural networks in general)
- The latent space may have “holes”, i.e., regions that don't correspond to valid data.
- You can't sample from it to generate new data.
- **VAE Key Idea**
  - Instead of encoding to a point, encode to a **probability distribution.** The encoder outputs **parameters of a Gaussian distribution**, and we sample from it

Variational Autoencoders

Dimensionality Reduction using Deep Learning